Patent Office
Canberra

I, LEANNE MYNOTT, TEAM LEADER EXAMINATION SUPPORT AND SALES hereby certify that annexed is a true copy of the Provisional specification in connection with Application No. PQ 0560 for a patent by SILVERBROOK RESEARCH PTY LTD filed on 25 May 1999.

CERTIFIED COPY OF PRIORITY DOCUMENT

WITNESS my hand this
Nineteenth day of April 2000

LEANNE MYNOTT
TEAM LEADER EXAMINATION
SUPPORT AND SALES

AUSTRALIA

Patents Act 1990

PROVISIONAL SPECIFICATION

Applicant(s):

SILVERBROOK RESEARCH PTY LTD

Invention Title:

A METHOD AND APPARATUS (PPRINT01)

The invention is described in the following statement:

A METHOD AND APPARATUS (PPRINT01)

Field of the Invention

The present invention relates to the field of print systems.

Background of the Invention

The present invention is directed to providing an extremely compact and versatile printing system utilizing an ink jet printing technology.

Summary of the Invention

It is an object of the present invention to provide a compact versatile printing system.

In accordance with a first aspect of the present invention, there is provided a pen shaped printing system comprising: a pen shaped module having a slot defined therein; a pagewidth ink jet printhead formed within the pen shaped module; a series of fluid supply reservoirs attached to the printhead for the supply of printer fluid; wherein print media for printing is inserted through the slot and an image to be printed is printed by the printhead.

The system can further include a detachable ink supply cartridge having the series of fluid supply reservoirs formed therein. The printhead can be a multicolor printhead and the detachable ink supply cartridge preferably can include a series of chambers, one for each color of ink. The pagewidth printhead can comprise a spaced apart series of ink ejection nozzles . The detachable ink supply cartridge can comprise a series of spaced apart longitudinal columar cavities filled with ink.

The system can further comprise a detachable imaging source module for the creation of images to be printed out by the printhead, the image being transferred to the pen shaped module for printout. The imaging source module can be cylindrically shaped. The detachable imaging source module can comprise a camera. The camera can comprise a swivel mounted image capture unit having a cylindrical shape in a closed position. The image capture unit preferably can include a viewfinder. The camera preferably can include a capping mechanism which forms a clip for attachment to a shirt pocket or the like.

The system can further comprise a flash module of a cylindrical shape for interconnecting with the detachable imaging source module. Further, there can be provided a detachable timer module of a cylindrical shape which activates the camera after a predetermined time. The timer module can activate the camera module at predetermined time intervals. The images captured by the timer are preferably stored in a separately detachable memory module.

The system can further comprise a computer network communication unit for interconnection with a computer network. The module can include a universal serial bus interconnector at one end and a cylindrically shaped interconnector at a second end.

The detachable imaging source module preferably can include a memory module adapted to store multiple images for print out on demand by the pen shaped printer. The detachable imaging source module preferably can include a display indicating a current image number, controls for altering the current image number and out and in buttons for forwarding images to and from the

detachable imaging source module. The images are preferably stored in a compressed form.

The detachable imaging source module can alternatively comprise an effects module for applying a filtering effect to the image to be printed out. The filtering effect preferably can include the placement of clip arts over portions of the image to be printed out or filtering the image to be printed. Examples of effects include effects relating to a constant theme, image warps, color mapping, or tiling an image. The filtering effects can include a series of clip arts directed to a predetermined topic or genre.

The pen shaped module preferably can include gender profiled ends and the printer further can also include a detachable gender changing module adapted to have two ends plugged into it of the same gender.

There can also be provided a card dispenser module coupled to the printer module for automatic dispensing of cards to be printed by the printer module. The printer module can be snap fitted into a cavity in the card dispenser module.

The detachable imaging source module can alternatively comprise a pager device or a sensor device or a detachable interchangeable nib module. The nib module can be an ink pen, a pencil, a stamp or a laser pointer. The nib module can be detached to provide access to an aperture in the pen shaped module containing a detachable battery device.

In accordance with a further aspect of the present invention, there is provided a pen shaped camera comprising: a cylindrically shaped camera module including an image sensor for capturing an image; a cylindrically shaped printer module having a slot defined therein; a pagewidth printhead formed within the cylindrically shaped printer module; a series of fluid supply reservoirs attached to the printhead for the supply of printer fluid; wherein print media for printing is inserted through the slot and an image to be printed is printed by the printhead. The camera module preferably can include memory storage facilities for capturing multiple images for printing out by the printhead.

Description Preferred in Other Embodiments

The preferred embodiment of the present invention is as set out in the attached appendix A which provides for a detailed description of the implementation of an information distribution and printing system denoted PENPRINT.

The ink jet technology utilized can be a pagewidth printing system as disclosed in the attached appendix B and denoted MEMJET.

It would be appreciated by a person skilled in the art that numerous variations and/or modifications may be made to the present invention as shown in the specific embodiments in the attached documentation without departing from the spirit or scope of the invention as broadly described in the attached appendices. The present embodiments are, therefore, to be considered in all respects to be illustrative and not restrictive.

A METHOD AND APPARATUS (PPRINT01)

Field of the Invention

The present invention relates to the field of print systems.

Background of the Invention

The present invention is directed to providing an extremely compact and versatile printing system utilizing an ink jet printing technology.

Summary of the Invention

It is an object of the present invention to provide a compact versatile printing system.

In accordance with a first aspect of the present invention, there is provided a pen shaped printing system comprising: a pen shaped module having a slot defined therein; a pagewidth ink jet printhead formed within the pen shaped module; a series of fluid supply reservoirs attached to the printhead for the supply of printer fluid; wherein print media for printing is inserted through the slot and an image to be printed is printed by the printhead.

The system can further include a detachable ink supply cartridge having the series of fluid supply reservoirs formed therein. The printhead can be a multicolor printhead and the detachable ink supply cartridge preferably can include a series of chambers, one for each color of ink. The pagewidth printhead can comprise a spaced apart series of ink ejection nozzles . The detachable ink supply cartridge can comprise a series of spaced apart longitudinal columar cavities filled with ink.

The system can further comprise a detachable imaging source module for the creation of images to be printed out by the printhead, the image being transferred to the pen shaped module for printout. The imaging source module can be cylindrically shaped. The detachable imaging source module can comprise a camera. The camera can comprise a swivel mounted image capture unit having a cylindrical shape in a closed position. The image capture unit preferably can include a viewfinder. The camera preferably can include a capping mechanism which forms a clip for attachment to a shirt pocket or the like.

The system can further comprise a flash module of a cylindrical shape for interconnecting with the detachable imaging source module. Further, there can be provided a detachable timer module of a cylindrical shape which activates the camera after a predetermined time. The timer module can activate the camera module at predetermined time intervals. The images captured by the timer are preferably stored in a separately detachable memory module.

The system can further comprise a computer network communication unit for interconnection with a computer network. The module can include a universal serial bus interconnector at one end and a cylindrically shaped interconnector at a second end.

The detachable imaging source module preferably can include a memory module adapted to store multiple images for print out on demand by the pen shaped printer. The detachable imaging source module preferably can include a display indicating a current image number, controls for altering the current image number and out and in buttons for forwarding images to and from the

detachable imaging source module. The images are preferably stored in a compressed form.

The detachable imaging source module can alternatively comprise an effects module for applying a filtering effect to the image to be printed out. The filtering effect preferably can include the placement of clip arts over portions of the image to be printed out or filtering the image to be printed. Examples of effects include effects relating to a constant theme, image warps, color mapping, or tiling an image. The filtering effects can include a series of clip arts directed to a predetermined topic or genre.

The pen shaped module preferably can include gender profiled ends and the printer further can also include a detachable gender changing module adapted to have two ends plugged into it of the same gender.

There can also be provided a card dispenser module coupled to the printer module for automatic dispensing of cards to be printed by the printer module. The printer module can be snap fitted into a cavity in the card dispenser module.

The detachable imaging source module can alternatively comprise a pager device or a sensor device or a detachable interchangeable nib module. The nib module can be an ink pen, a pencil, a stamp or a laser pointer. The nib module can be detached to provide access to an aperture in the pen shaped module containing a detachable battery device.

In accordance with a further aspect of the present invention, there is provided a pen shaped camera comprising: a cylindrically shaped camera module including an image sensor for capturing an image; a cylindrically shaped printer module having a slot defined therein; a pagewidth printhead formed within the cylindrically shaped printer module; a series of fluid supply reservoirs attached to the printhead for the supply of printer fluid; wherein print media for printing is inserted through the slot and an image to be printed is printed by the printhead. The camera module preferably can include memory storage facilities for capturing multiple images for printing out by the printhead.

Description Preferred in Other Embodiments

The preferred embodiment of the present invention is as set out in the attached appendix A which provides for a detailed description of the implementation of an information distribution and printing system denoted PENPRINT.

The ink jet technology utilized can be a pagewidth printing system as disclosed in the attached appendix B and denoted MEMJET.

It would be appreciated by a person skilled in the art that numerous variations and/or modifications may be made to the present invention as shown in the specific embodiments in the attached documentation without departing from the spirit or scope of the invention as broadly described in the attached appendices. The present embodiments are, therefore, to be considered in all respects to be illustrative and not restrictive.

We Claim:

1. A pen shaped printing system comprising:

a pen shaped module having a slot defined therein;

a pagewidth ink jet printhead formed within the pen shaped module;

a series of fluid supply reservoirs attached to said printhead for the supply of printer fluid;

wherein print media for printing is inserted through said slot and an image to be printed is printed by said printhead.

2. A pen shaped printing system as claimed in claim 1 further comprising:

a detachable ink supply cartridge having said series of fluid supply reservoirs formed therein.

3. A pen shaped printing system as claimed in claim 2 wherein said printhead is a multicolor printhead and said detachable ink supply cartridge includes a series of chambers, one for each color of ink.

4. A pen shaped printing system as claimed in claim 3 wherein said pagewidth printhead comprises a spaced apart series of ink ejection nozzles .

5. A pen shaped printing system as claimed in claim 3 wherein said detachable ink supply cartridge wherein said series of chambers comprise a series of spaced apart longitudinal columar cavities filled with ink.

Detachable modules

6. A pen shaped printing system as claimed in any previous claim further comprising:

a detachable imaging source module for the creation of images to be printed out by said printhead, said image being transferred to said pen shaped module for printout.

7. A pen shaped printing system as claimed in claim 6 wherein said imaging source module is cylindrically shaped.

8. A pen shaped printing system as claimed in claim 6 wherein said detachable imaging source module comprises a camera.

9. A pen shaped printing system as claimed in claim 8 wherein said camera comprises a swivel mounted image capture unit having a cylindrical shape in a closed position.

10. A pen shaped printing system as claimed in claim 9 wherein said image capture unit includes a viewfinder.

11. A pen shaped printing system as claimed in claim 8 wherein said camera includes a capping mechanism which forms a clip for attachment to a shirt pocket or the like.

12. A pen shaped printing system as claimed in claim 6 further comprising a detachable flash module of a cylindrical shape for interconnecting with said detachable imaging source module.

13. A pen shaped printing system as claimed in claim 12 wherein said detachable flash module is cylindrically shaped.

14. A pen shaped printing system as claimed in claim 6 further comprising a detachable timer module of a cylindrical shape which activates said camera after a predetermined time.

15. A pen shaped printing system as claimed in claim 6 wherein said timer module activates said camera module at predetermined time intervals.

16. A pen shaped printing system as claimed in claim 15 wherein the images captured by said timer are stored in a separately detachable memory module.

USB connector

17. A pen shaped printing system as claimed in claim 6 wherein said detachable imaging source module comprises a computer network communication unit for interconnection with a computer network.

18. A pen shaped printing system as claimed in claim 17 wherein said detachable imaging source module comprises a universal serial bus interconnector.

19. A pen shaped printing system as claimed in claim 17 wherein said detachable imaging source module includes a USB connector at one end and a cylindrically shaped interconnector at a second end.

memory module

20. A pen shaped printing system as claimed in claim 6 wherein said detachable imaging source module includes a memory module adapted to store multiple images for print out on demand by said pen shaped printer.

21. A pen shaped printing system as claimed in claim 20 wherein said detachable imaging source module includes a display indicating a current image number, controls for altering the current image number and out and in buttons for forwarding images to and from said detachable imaging source module.

22. A pen shaped printing system as claimed in claim 20 wherein said images are stored in a compressed form.

23. A pen shaped printing system as claimed in claim 6 wherein said detachable imaging source module comprises an effects module for applying a filtering effect to the image to be printed out.

24. A pen shaped printing system as claimed in claim 23 wherein said filtering effect includes the placement of clip arts over portions of said image to be printed out.

25. A pen shaped printing system as claimed in claim 23 wherein said filtering effect includes filtering said image to be printed.

26. A pen shaped printing system as claimed in claim 23 wherein said filtering effect relate to a constant theme.

27. A pen shaped printing system as claimed in claim 23 wherein said filtering effect includes image warps.

28. A pen shaped printing system as claimed in claim 23 wherein said filtering effect includes color mapping.

29. A pen shaped printing system as claimed in claim 23 wherein said filtering effect includes tiling an image.

30. A pen shaped printing system as claimed in claim 23 wherein said filtering effects include a series of clip arts directed to a predetermined topic or genre.

31. A pen shaped printing system as claimed in claim 1 wherein said pen shaped module includes gender profiled ends and said printer further comprises:

a detachable gender changing module adapted to have two ends plugged into it of the same gender.

32. A pen shaped printing system as claimed in any previous claim further comprising:

a card dispenser module coupled to said printer module for automatic dispensing of cards to be printed by said printer module.

33. A pen shaped printing system as claimed in claim 32 wherein said printer module is snap fitted into a cavity in said card dispenser module.

34. A pen shaped printing system as claimed in claim 4 wherein said detachable imaging source module comprises a pager device.

35. A pen shaped printing system as claimed in claim 4 wherein said detachable imaging source module comprises a sensor device.

36. A pen shaped printing system as claimed in claim 1 further comprising:

a detachable interchangeable nib module.

37. A pen shaped printing system as claimed in claim 36 wherein said nib module includes an ink pen.

38. A pen shaped printing system as claimed in claim 36 wherein said nib module includes a pencil.

39. A pen shaped printing system as claimed in claim 36 wherein said nib module includes a stamp.

40. A pen shaped printing system as claimed in claim 36 wherein said nib module includes a laser pointer.

41. A pen shaped printing system as claimed in claim 36 wherein said nib module is detached to provide access to an aperture in said pen shaped module containing a detachable battery device.

42. A pen shaped camera comprising:

a cylindrically shaped camera module including an image sensor for capturing an image;

a cylindrically shaped printer module having a slot defined therein;

a pagewidth printhead formed within the cylindrically shaped printer module;

a series of fluid supply reservoirs attached to said printhead for the supply of printer fluid;

wherein print media for printing is inserted through said slot and an image to be printed is printed by said printhead.

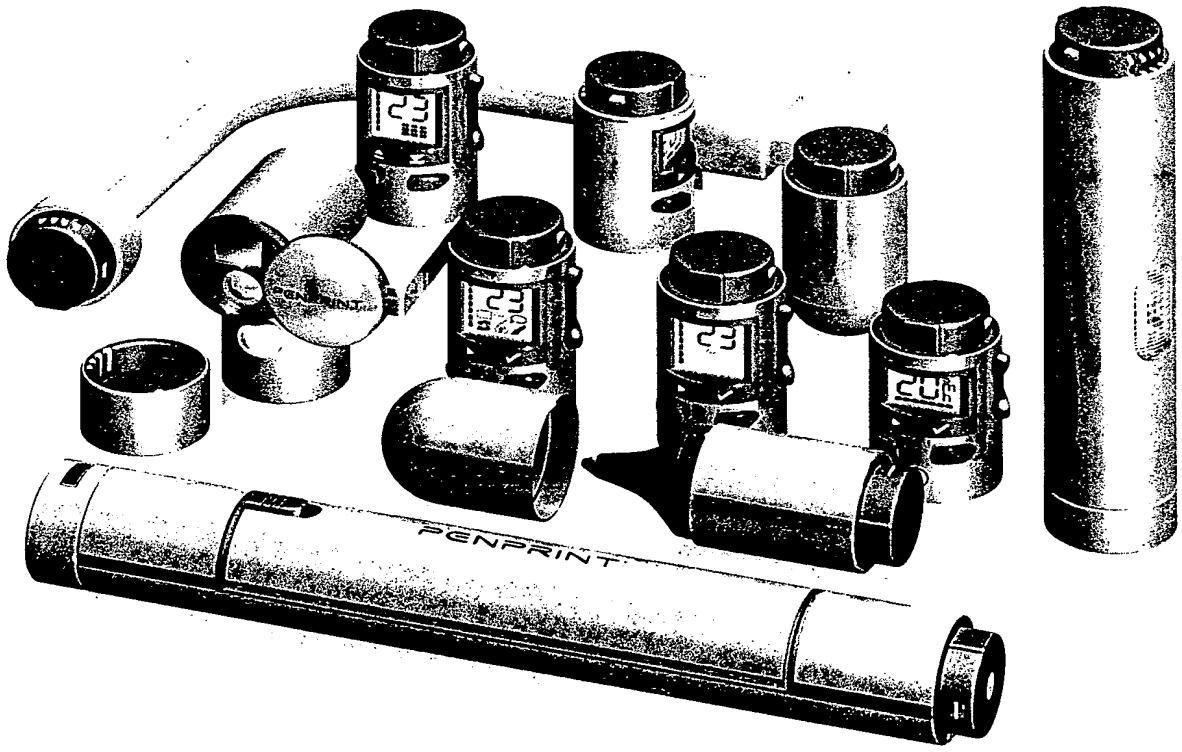
43. A pen shaped camera as claimed in claim 42 wherein said camera module includes memory storage facilities for capturing multiple images for printing out by said printhead.

44. A pen shaped printing system substantially as hereinbefore described with reference to the accompanying appendices.

45. A series of detachable modules for a pen shaped printing system substantially as hereinbefore described with reference to the accompanying appendices.

PenPrint
Product Design Description

version 1.0



Silverbrook Research Pty Ltd
393 Darling Street, Balmain
NSW 2041 Australia
Phone: +61 2 9818 6633
Fax: +61 2 9818 6711
Email: info@silverbrook.com.au

Document History

Version	Date	Author	Details
1.0	5 May 1999	Simon Walmsley Toby King Kia Silverbrook	Initial version.

Contents

PenPrint Overview 1
1 Introduction 2
1.1 Operational Overview 2
1.2 Document Summary 3
2 PenPrint System Model 4
2.1 Connectivity 4
2.2 Types of PenPrint Modules 4
2.2.1 Image Processing Modules 4
2.2.2 Housekeeping Modules 5
2.2.3 Isolated Modules 5
2.3 Basic PenPrint Configurations 5
2.3.1 Minimum Configuration 5
2.3.2 Minimum Configuration Plus Camera Module 6
2.3.3 Minimum Configuration Plus Memory Module 7
2.3.4 Minimum Configuration Plus USB Module 7
2.3.5 General Case 7
2.4 Standard Image Type 8
2.4.1 Image Attributes 8
2.4.2 Granularity of Access 9
2.5 User Interface 9
2.5.1 Physical Interface 9
2.5.2 Logical Interface 10
3 PenPrint Serial Bus 11
3.1 Features / Requirements 11
3.1.1 Reasonable Transfer Speed 11
3.1.2 Dynamic Attach / Detach 11
3.1.3 Auto Configuration 11
3.1.4 Expandability / Room for Growth 12
3.1.5 Low Cost 12
3.1.6 Low Power 12
3.2 Standard Module Functions 12

PenPrint Module Overview 14
4 Introduction 15
5 Printer Module 18
5.1 Overview 18
5.2 Appearance 18
5.3 Method of Operation 18
5.3.1 Printing Images 18
5.3.2 Replacing ink 20
5.3.3 Replacing battery 21
5.4 Computer Interface 21
5.5 Internals 22

6 Camera Module 24
6.1 Overview 24
6.2 Appearance 24
6.3 Method of Operation 24
6.4 Computer Interface 25
6.5 Internals 26
7 Memory Module 28
7.1 Overview 28
7.2 Appearance 28
7.3 Method of Operation 28
7.4 Computer Interface 29
7.5 Internals 29
7.5.1 Storage of Images 29
7.5.2 ASIC 30
8 USB Module 31
8.1 Overview 31
8.2 Appearance 31
8.3 Method of Operation 31
9 Flash Module 32
9.1 Overview 32
9.2 Appearance 32
9.3 Method of Operation 32
9.4 Internals 33
10 Timer Module 34
10.1 Overview 34
10.2 Appearance 34
10.3 Method of Operation 34
10.4 Internals 35
11 Laser Pointer Module 36
11.1 Overview 36
11.2 Appearance 36
11.3 Method of Operation 36
11.4 Internals 36
12 Effects Module 37
12.1 Overview 37
12.2 Appearance 37
12.3 Method of Operation 37
12.4 Effect Types 38
12.4.1 Borders 39
12.4.2 Clip-art 40
12.4.3 Captions 41
12.4.4 Warps 42
12.4.5 Color Changes 43
12.4.6 Painting Styles 44
12.5 Internals 45

13 Character Module 46
13.1 Overview 46
13.2 Appearance 46
13.3 Licensing Issues 46
13.4 Technical Issues 46
13.5 Method of Operation 47
14 Gender Changer Module 48
14.1 Overview 48
14.2 Appearance 48
14.3 Method of Operation 48
15 Pen Module 49
15.1 Overview 49
15.2 Appearance 49
15.3 Method of Operation 49

Printer Module 50
16 Introduction 51
16.1 Overview 51
16.2 Appearance 51
16.3 Method of Operation 51
16.3.1 Printing Images 51
16.3.2 Replacing ink 53
16.3.3 Replacing battery 54
16.4 Computer Interface 54
16.5 Internals 54
17 Image Processing Requirements 57
17.1 Constraints 57
17.1.1 Timing 57
17.2 Image Print Chain 58
17.2.1 Convert from L*a*b* to CMY 58
17.2.2 Up Interpolate 59
17.2.3 Halftone 60
17.2.4 Reformat for Printer 61
18 PenPrint Central Processor (PPCP) 62
18.1 CPU Core and Memory 62
18.1.1 CPU Core 62
18.1.2 Program ROM 63
18.1.3 Program RAM 63
18.1.4 CPU Memory Decoder 63
18.2 Communication Interfaces 63
18.2.1 PenPrint Serial Port Interface 63
18.2.2 QA Chip Serial Interfaces 63
18.2.3 Parallel Interface 64
18.2.4 JTAG Interface 65
18.3 Image RAM 65
18.4 Image Access Unit 65
18.5 Printhead Interface 66

18.5.1 LineSyncGen Unit 67
18.5.2 Memjet Interface 68
18.5.3 Print Generator Unit 74

Memjet Printhead 87
19 Introduction 88
20 Memjet Segment Structure 89
20.1 Grouping of Nozzles Within a Segment 89
20.1.1 10 Nozzles Make a Pod 89
20.1.2 1 Pod of Each Color Makes a Chromapod 90
20.1.3 5 Chromapods Make a Podgroup 91
20.1.4 2 Podgroups Make a Phasegroup 91
20.1.5 2 Phasegroups Make a Firegroup 91
20.1.6 Nozzle Grouping Summary 92
20.2 Load and Print Cycles 93
20.2.1 Load Cycle 93
20.2.2 Print Cycle 94
20.3 Feedback from a Segment 97
20.4 Preheat Cycle 97
20.5 Cleaning Cycle 97
20.6 Printhead Interface Summary 98
21 Making Memjet Printheads out of Segments 99
21.1 Loading Considerations 100
21.2 Printing Considerations 101
21.3 Feedback Considerations 101
21.4 Printhead Connection Summary 102

Camera Module 103
22 Introduction 104
22.1 Overview 104
22.2 Appearance 104
22.3 Method of Operation 104
22.4 Computer Interface 105
22.5 Internals 106
23 Image Processing Requirements 108
23.1 Image Capture Chain 109
23.1.1 Image Sensor 109
23.1.2 Linearize RGB 109
23.1.3 Planarize RGB 110
23.1.4 Stored Image 111
23.2 Image Enhancement Chain 111
23.2.1 Input Image 112
23.2.2 Gather Statistics 112
23.2.3 White Balance and Range Expansion 114
23.2.4 Resample 115
23.2.5 Convert to L*a*b* 119
23.2.6 Sharpen 120

24 PenPrint Camera Image Processor (PPCIP) 122
24.1 High Level Internal Overview 122
24.2 CPU Core and Memory 122
24.2.1 CPU Core 122
24.2.2 Program ROM 123
24.2.3 Program RAM 123
24.2.4 CPU Memory Decoder 123
24.3 Communication Interfaces 123
24.3.1 PenPrint Serial Bus Interface 123
24.3.2 Parallel Interface 123
24.3.3 JTAG Interface 123
24.4 Image RAM 124
24.5 Image Capture Unit 124
24.5.1 Image Sensor Interface 124
24.5.2 Lookup Table 125
24.5.3 State Machine 125
24.5.4 Registers 126
24.6 Image Histogram Unit 126
24.6.1 Histogram RAM 127
24.6.2 State Machine and Registers 127
24.7 Image Enhancement Unit 127
24.7.1 Buffer 1 130
24.7.2 Buffer 2 131
24.7.3 Buffer 3 131
24.7.4 Buffer 4 132
24.7.5 White Balance and Range Expansion 133
24.7.6 Resample 138
24.7.7 Convert to L*a*b* 150
24.7.8 Sharpen 153

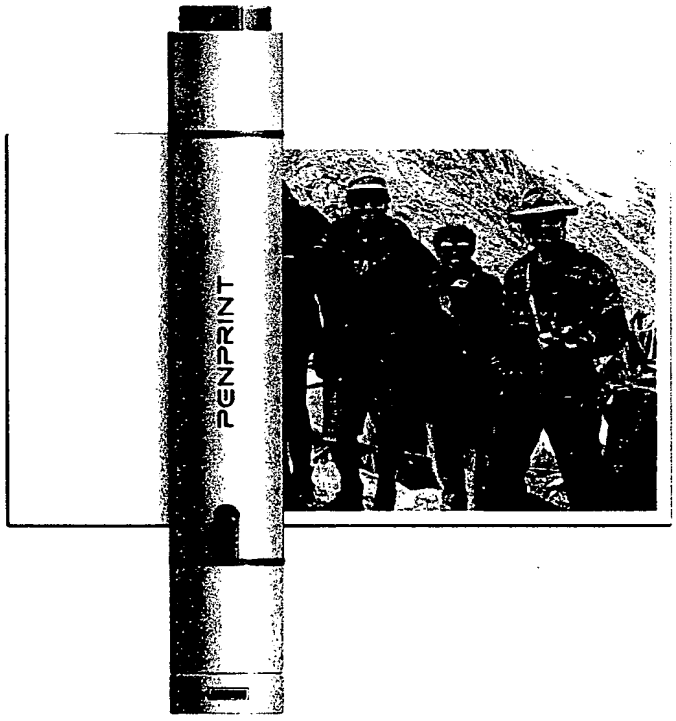
Effects Module 155

25 Effects Module 156
25.1 Overview 156
25.2 Appearance 156
25.3 Method of Operation 156
25.4 Effect Types 157
25.4.1 Borders 158
25.4.2 Clip-art 159
25.4.3 Captions 160
25.4.4 Warps 161
25.4.5 Color Changes 162
25.4.6 Painting Styles 163
25.5 Internals 164

26 Effects Module Central Processor 165
26.1 Implementation of Effects using EMCP 165
26.1.1 Compositing 165
26.1.2 Convolve 166
26.1.3 Color Substitution 167
26.1.4 Construction of Image Pyramid 168
26.1.5 Warping an Image 169

References 173
27 References 174

PenPrint Overview



1 Introduction

PenPrint is a high-performance color imaging system ***in a pen***. It features a central printer module capable of photographic-quality image reproduction via a 2 inch Memjet [6] printhead which produces 1600 dots per inch (dpi) bi-level CMY (Cyan, Magenta, Yellow). PenPrint prints a single business card sized (85mm × 55mm) image in 1 second. Figure 1 shows a *life-size* image of the basic printer module printing a card.

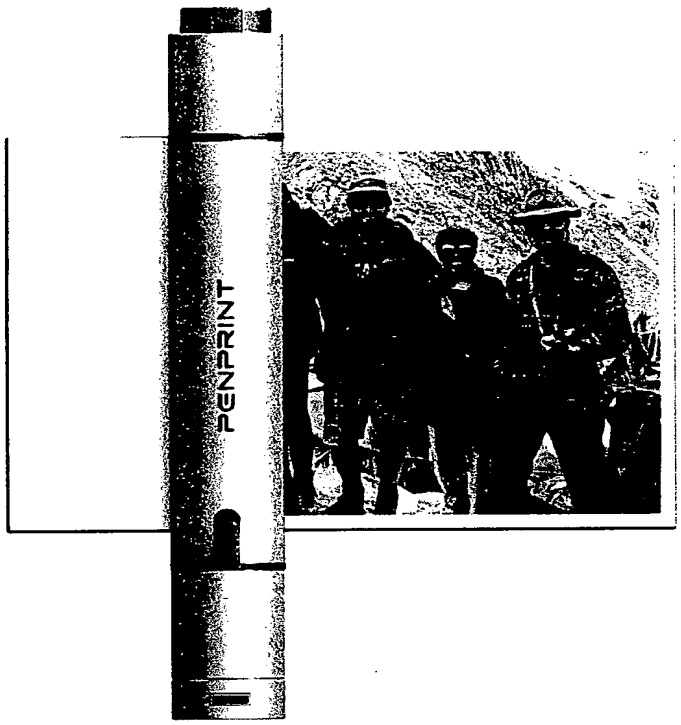


Figure 1. Basic Printer Module Printing an Image (life size)

Additional modules for image capture, mass image storage, image manipulation, USB connection to PCs etc. all attach to the printer module in an extendable, interchangeable way.

1.1 Operational Overview

PenPrint reproduces graphics directly using continuous-tone (contone) images and graphics using dithered bi-level CMY. For practical purposes, PenPrint supports a contone resolution of 267 pixels per inch (ppi).

Images are captured by the PenPrint Camera Module or are generated on a computer and downloaded via the PenPrint USB Module. Images are printed out by inserting a business card into the PenPrint Printer Module.

Individual modules can be attached and detached from a PenPrint configuration to allow a user-definable solution to business-card sized printing. Images can also be transferred from one PenPrint to another without the use of a secondary computer system. Modules have a minimal user-interface to allow straightforward interaction.

1.2 DOCUMENT SUMMARY

This document is broken up into parts, each containing a number of chapters. This part is a summary product specification for PenPrint. The second part of the document (page 14) provides a summary for all of the PenPrint modules. Subsequent parts provide further detail for each PenPrint module, starting with the central PenPrint Printer Module (page 50).

2 PenPrint System Model

2.1 CONNECTIVITY

A PenPrint system configuration simply consists of a number of PenPrint modules connected together. Each PenPrint module has a function that contributes to the overall functionality of the particular PenPrint configuration. Each PenPrint module is typically shaped like part of a pen, physically connecting with other PenPrint modules to form the complete pen-shaped device. The length of the PenPrint device depends on the number and type of PenPrint modules connected. The functionality of a PenPrint configuration depends on the PenPrint modules in the given configuration.

The PenPrint modules connect both physically and logically. The physical connection allows modules to be connected in any order, and the logical connection is taken care of by the PenPrint Serial Bus - a bus that provides power, allows the modules to self configure and provides for the transfer of data.

In terms of physical connection, most PenPrint modules consist of a central body, a male connector at one end, and a female connector at the other. Since most modules have both a male and female connector, the modules can typically be connected in any order. Certain modules only have a male or a female connector, but this is determined by the function of the module. Gender changing modules allow these single-connector modules to be connected at either end of a given PenPrint configuration.

A 4 wire physical connection between all the PenPrint modules provides the logical connection between them in the form of the PenPrint Serial Bus. The PenPrint Serial Bus provides power to each module, and provides the means by which data is transferred between modules. Importantly, the PenPrint Serial Bus and accompanying protocol provides the means by which a PenPrint system auto-configures, reducing the user-interface burden on the end-user.

2.2 TYPES OF PENPRINT MODULES

PenPrint modules can be grouped into three types:

- image processing modules
- housekeeping modules
- isolated modules

Although housekeeping modules and isolated modules are useful components in a PenPrint system, they are extras in a system dedicated to image processing and photographic manipulation. Each of the PenPrint modules is described in detail in subsequent sections within this document. Life size (1:1) illustrations of the PenPrint modules can be found in Figure 6, and example configurations produced by connecting various modules together can be found in Figure 7.

2.2.1 Image Processing Modules

Image processing modules are primarily what sets PenPrint modules apart from other pen-like devices. Image processing modules capture, print, store or manipulate photographic images.

Examples include the Printer Module, Camera Module, and Memory Module.

2.2.2 Housekeeping Modules

Housekeeping modules provide services to other modules or extended functionality to other modules.

Examples include the Gender Changer Module, and Timer Module.

2.2.3 Isolated Modules

Isolated modules are those that attach to a PenPrint system but are completely independent of any other module. They do not necessarily require power, and may even provide their own power. Isolated Modules are defined because the functionality they provide is typically incorporated into other pen-like devices.

Examples include the Pen Module and Laser Pointer Module.

2.3 Basic PenPrint Configurations

This section describes some basic PenPrint configurations. Life size (1:1) illustrations of example configurations can be found in Figure 7 on page 17.

2.3.1 Minimum Configuration

In the sense that a minimum configuration PenPrint system must be able to print out photos, a minimum PenPrint configuration contains at least a Printer Module. The Printer Module holds a single photographic image that can be printed out via its Memjet printer. It also contains the 3V battery required to power the PenPrint system. Figure 2 shows the basic printer Module.

Figure 2. Basic Printer Module

In this minimum configuration, the user is only able to print out photos. Each time a user inserts a business card into the slot in the Printer Module, the image in the Printer Module is printed onto the card. The same image is printed each time a business card is inserted into the printer. In this minimum configuration there is no way for a user to change the image that is printed[1]. At this point we have not described *how* the image has been placed

1. There is provision for a user to *erase* the image from the Printer Module so that subsequent images are simply blank images. However the Printer Module itself does not store multiple images.

in the Printer Module, only that it *is* there. The optional card dispenser can be used to feed cards into the Printer Module with a minimum of fuss.

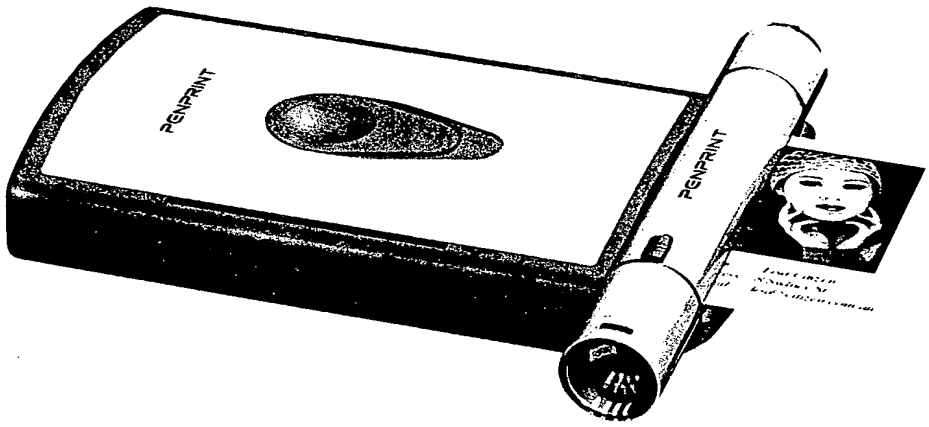


Figure 3. Printer Module with Optional Card Dispenser

2.3.2 Minimum Configuration Plus Camera Module

By connecting a Camera Module to the minimum configuration PenPrint system the user now has an instant printing digital camera in a pen. The Camera Module provides the mechanism for capturing images and the Printer Module provides the mechanism for printing them out. The battery in the Printer Module provides power for both the camera and the printer.

When the user presses the "Take" button on the Camera Module, the image is captured by the camera and transferred to the Printer Module. Each time a business card is inserted into the printer the captured image is printed out. If the user presses "Take" on the Camera Module again, the old image in the Printer Module is replaced by the new image.

If the Camera Module is subsequently detached from the PenPrint system, the captured image remains in the Printer Module, and can be printed out as many times as desired. The Camera Module then, is simply there to capture images to be placed in the Printer Module.

Figure 4 shows the configuration of Printer Module plus Camera Module.

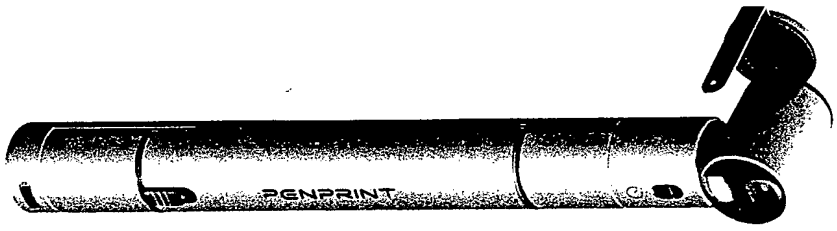


Figure 4. Printer Module plus Camera Module

2.3.3 Minimum Configuration Plus Memory Module

By connecting a Memory Module to the minimum configuration PenPrint system, the user has the ability to transfer images between the Printer Module and a storage area contained in the Memory Module. The user selects the image number on the Memory Module, and then either sends that image to the Printer Module (replacing whatever image was already stored there), or brings the current image from the Printer Module to the specified image number in the Memory Module. The Memory Module also provides a way of sending sets of thumbnail images to the Printer Module.

Multiple Memory Modules can be included in a given system, simply extending the number of images that can be stored. A given Memory Module can be disconnected from one PenPrint system and connected to another for subsequent image printing.

If a Camera Module is also attached to a Memory Module/Printer Module PenPrint system, the user can "Take" an image with the Camera Module, then transfer it to the specified image number in the Memory Module. The captured images can then be printed out in any order.

Figure 5 shows the configuration of Printer Module plus Camera and Memory Modules.

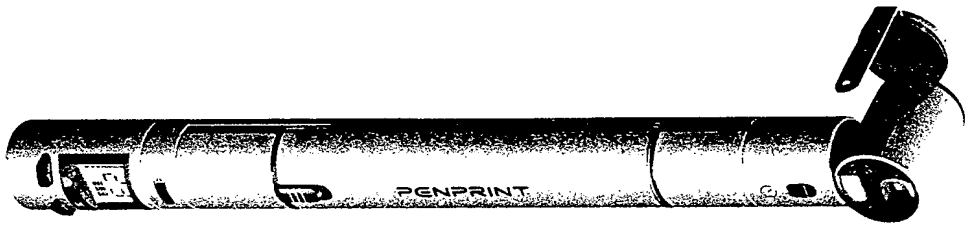


Figure 5. Printer Module plus Camera and Memory Modules

2.3.4 Minimum Configuration Plus USB Module

By connecting a USB Module to the minimum configuration PenPrint system, the user gains the ability to transfer images between the PC and the PenPrint system. The USB Module makes the Printer Module and any Memory Modules visible to an external computer system. This allows the download or uploading of images.

The USB Module also allows computer control of any connected PenPrint modules, such as the Camera Module.

2.3.5 General Case

In the general case, the Printer Module holds the "current" image, and the other modules function with respect to this central repository of the current image. The PenPrint Printer Module is therefore the central location for image interchange in the PenPrint system, and the Printer Module provides a service to other modules as specified by user interaction.

A given module may act as an image source. It therefore has the ability to transfer an image to the Printer Module. A different module may act as an image store. It therefore has the ability to read the image from the Printer Module. Some modules act as both image store and image source. These modules can both read images from and write images to the Printer Module's current image.

2.4 STANDARD IMAGE TYPE

The image processing modules in a PenPrint system all operate on a standard image type.

2.4.1 Image Attributes

The standard image type has a single conceptual definition. At this point we are not concerned about *how* the image is stored (row-wise or column-wise, interleaved or planer etc.), but rather the concept of *what* an image is in the PenPrint system.

2.4.1.1 Resolution

The image definition is derived from the physical attributes of the Memjet printhead used in the PenPrint Printer Module. The Memjet printhead is 2 inches wide and prints at 1600dpi in cyan, magenta and yellow bi-level dots (the physical attributes of the Memjet printhead are defined in Section 19 on page 88). Consequently a printed image from a PenPrint system is 3200 bi-level dots wide.

The PenPrint system prints on business card sized pages (85mm × 55mm). Since the printhead is 2 inches wide, the business cards are printed such that 1 line of dots is 2 inches. 2 inches is 50.8mm, leaving a 2mm edge on a standard business-card sized page. The length of the image is derived from the same card size with a 2mm edge. Consequently the printed image length is 81mm, which equals 5100 1600dpi dots. The printed area of a page is therefore **81mm × 51mm**, or **5100 × 3200 dots**.

To obtain an integral contone to bi-level ratio we choose a contone resolution of 267 ppi (pixels per inch). This yields a contone CMY page size of **850 × 534**[1], and a contone to bi-level ratio of 1:6 in each dimension. This ratio of 1:6 provides no perceived loss of quality since the output image is bi-level.

2.4.1.2 Color

The Memjet printhead prints dots in cyan, magenta, and yellow ink. The final output to the printed page must therefore be in the gamut of the printhead and take the attributes of the inks into account. It would at first seem reasonable to use the CMY color space to represent images.

However, the printer's CMY color space does not have a linear response. This is definitely true of pigmented inks, and partially true for dye-based inks. The individual color profile of a particular device (input and output) can vary considerably. Image capture devices (such as digital cameras) typically work in RGB (red green blue) color space, and each sensor will have its own color response characteristics.

Consequently, to allow for accurate conversion, as well as to allow for future image sensors, inks, and printers, **the CIE L*a*b* color model [1] is used for PenPrint**. L*a*b* is well defined[2], perceptually linear, and is a superset of other traditional color spaces (such as CMY, RGB, and HSV).

1. 534 × 6 = 3204, which means that the last 4 dots in a given line are not printed. We are limited by the Memjet printhead having *exactly* 3200 dots of each color.
2. Notation L* is a measure of lightness, and varies from perfect white (100%) to absolute black (0%); a* measures the red-green color balance; b* measures the yellow-blue color balance. See [1] for full details.

Any PenPrint Printer Module must therefore be capable of converting L*a*b* images to the particular peculiarities of its CMY color space. However, since PenPrint systems allow for connectivity to PCs, it is quite reasonable to also allow highly accurate color matching between screen and printer to be performed on the PC. However the printer driver or PC program must output L*a*b*.

Each pixel of a PenPrint image is therefore represented by 24 bits: 8 bits each of L*, a*, and b*. The total image size is therefore 1,361,700 bytes (850 × 534 × 3).

2.4.2 Granularity of Access

Each image processing PenPrint module must be able to access the image stored in the PenPrint Printer Module. The access is either to read the image from the Printer Module, or to write a new image to the Printer Module.

The communications protocol for image access to the Printer Module provides a choice of internal image organization. Images can be accessed either as 850 × 534 or as 534 × 850. They can also be accessed in interleaved or planar format. When accessed as interleaved, each pixel in the image is read or written as 24 bits: 8 bits each of L*, a*, b*. When accessed as planar, each of the color planes can be read or written independently. The entire image of L* pixels, a* pixels or b* pixels can be read or written at a time.

2.5 USER INTERFACE

There are two user interfaces to consider:

- the physical interface for PenPrint operations
- the logical interface for PenPrint image transfer

A minimal user interface is desirable to maximize usability in both cases.

2.5.1 Physical Interface

A PenPrint configuration is constructed by physically connecting multiple PenPrint modules together via the male/female bayonet connectors. There is no specific "on/off" switch to take account of. The PenPrint configuration runs in standby mode and allows hot attachment/detachment of modules.

The PenPrint system works on the concept of a *current image* that is located in the Printer Module. Whenever a card is inserted into the Printer Module, the *current* image is printed out. The insertion of multiple cards into the Printer Module causes multiple copies of the same image to be printed.

The remainder of the physical user interface is concerned with instructing a given Module to do some task. In the case of Image Processing Modules the task is usually with respect to the current image in the Printer Module. The user is able to change the *current* image by instructing a given PenPrint module to send its image to the Printer Module. The method for selecting *which* image to send depends on the module. The Camera Module, for example, has a single ***Take*** button. Whenever the ***Take*** button is pressed, an image is captured by the camera and transferred to become the current image in the Printer Module. The Memory Module is slightly more complex: two buttons ***Out*** and ***In*** send an image out to the Printer Module, or take an image in from the Printer Module. An additional button on the Memory Module allows the user to cycle through the desired image number.

2.5.2 Logical Interface

A computer attached to the PenPrint system via a USB Module sees the various images on the PenPrint device chain, and software (in the computer) allows images to be read from or written to the addresses within the PenPrint devices.

In the logical interface, the Printer Module's current image does not have to be the source or destination of each image transfer. Instead, the computer system becomes the effective current image, reading images from any of the PenPrint modules, and writing images to any of the PenPrint modules.

3 PenPrint Serial Bus

The PenPrint Serial Bus is the internal backbone bus for the PenPrint system. It provides power for the PenPrint modules, allows the various PenPrint modules to communicate with each other, allows images to be transferred between them, and provides the means for growing /expanding the products in the PenPrint range.

3.1 FEATURES / REQUIREMENTS

The following features are required by the PenPrint Serial Bus:

- Reasonable Transfer Speed
- Dynamic Attach / Detach
- Auto Configuration
- Expandability / Room for Growth
- Low Cost
- Low Power

The Universal Serial Bus (USB) specification caters for the requirements of PenPrint. USB is well defined and is easily integrated into a hardware architecture.

3.1.1 Reasonable Transfer Speed

The PenPrint Serial Bus requires a medium speed for transferring images. While not the high speed required for video, a reasonable speed for image transmission is required.

Although the maximum speed on USB is 12 MBits/sec, the maximum effective data transfer rate is 8MBits/sec due to protocol overhead and transmission redundancy. The time taken to transmit a complete image (850 × 534 L*a*b*) is therefore **1.36 seconds** (850 × 534 × 3 × 8 / 8,000,000).

3.1.2 Dynamic Attach / Detach

It is imperative that the attachment or detachment of a module from the PenPrint system does not adversely affect the module or the rest of the system. This is also necessary from an ease-of-use standpoint. Users of PenPrint are likely to attach modules and detach modules to reconfigure their system. The notion of individual on/off switches per module or special attachment procedures can be a large deterrent.

USB supports hot attachment as a design requirement.

3.1.3 Auto Configuration

The PenPrint modules must be self identifying, and self configuring (including bus termination). Users must not need to know anything about the electrical system. Users should simply need to connect the appropriate modules together and the PenPrint system will configure itself appropriately.

USB was specifically designed to allow auto configuration, and caters for self-identifying devices.

3.1.4 Expandability / Room for Growth

Any given PenPrint configuration consists of a number of PenPrint modules connected to one another. The exact number of PenPrint modules depends on the requirements of the user. In addition, the complete set of PenPrint modules is not fixed. While this document describes a number of PenPrint modules, it is by no means complete. The PenPrint serial bus should allow new PenPrint modules to be easily designed and added to the PenPrint family.

USB allows up to 127 physical devices to be connected at a given time (power requirements not withstanding). Since the devices are self identifying, new modules can be added to a PenPrint system as long as they conform to the PenPrint serial protocol.

3.1.5 Low Cost

The design effort for a serial connection between PenPrint modules should take advantage of existing serial interface solutions. Using existing well-defined solutions such as USB reduces design time, allows the use of pre-compiled cores, debugged protocols, and gives a large range of choice in terms of components. The choice of USB also allows the PenPrint to be easily integrated into a PC environment for image download and upload.

3.1.6 Low Power

PenPrint runs directly off an unregulated 3V power supply (a 3V battery). The PenPrint Serial Bus must also run off this power.

USB nominally runs at 5V. Under normal circumstances, voltage at a powered hub-port must be no less than 4.75Vdc, while voltage at a bus-powered hub must be no less than 4.40Vdc. However this is the case for a completely open system (which PenPrint is *not*). Consequently the USB used within PenPrint modules must be lower power, and therefore run off an unregulated 3V power supply. This is the only difference between regular USB and USB as used in PenPrint.

3.2 STANDARD MODULE FUNCTIONS

Each PenPrint module is visible on the PenPrint Serial Bus. Each module is self identifying and self-configuring using standard USB protocols.

Apart from the standard protocol functions (including identification), there are a number of functions that each PenPrint module must be capable of responding to. These are outlined in Table 1. Each module also has a number of module-specific functions.

Table 1. Basic PenPrint Module Functions

Name	Description
GetImageCounts()	Returns two counts - the number of images that can be read from the module, and the number of images that can be written to the module. This allows read only, write only, and virtual read only images.
GetCurrentImageNumber	If the module has a setting for the image number, this call returns the current image number.
GetImageAccessMethods	Returns two sets of 8 access bits. The first set represents the read access bits, and the second set represents the write access bits. See Table 2 for an interpretation of the bits.

Table 1. Basic PenPrint Module Functions

Name	Description
GetImage(N, Mode)	Returns image number N using the specified 8-bit access mode. See Table 3 for an interpretation of the access mode bits.
StoreImage(N, Mode)	Stores an image at address N using the specified 8-bit access mode. See Table 3 for an interpretation of the access mode bits.
TransferImage(N, Mode, Dest)	Transfers the image at address N using the specified 8-bit access mode to the serial device with id Dest. See Table 3 for an interpretation of the access mode bits.

The 8-bit mode returned by GetImageAccessMethods is interpreted as follows:

Table 2. 8-bit return code from GetImageAccessMethods

Bit	Interpretation
0	Access 850 × 534 permitted
1	Access 534 × 850 permitted
2	Interleaved L*a*b* permitted
3	Planar L*, a*, b* permitted
4-7	Reserved, and 0

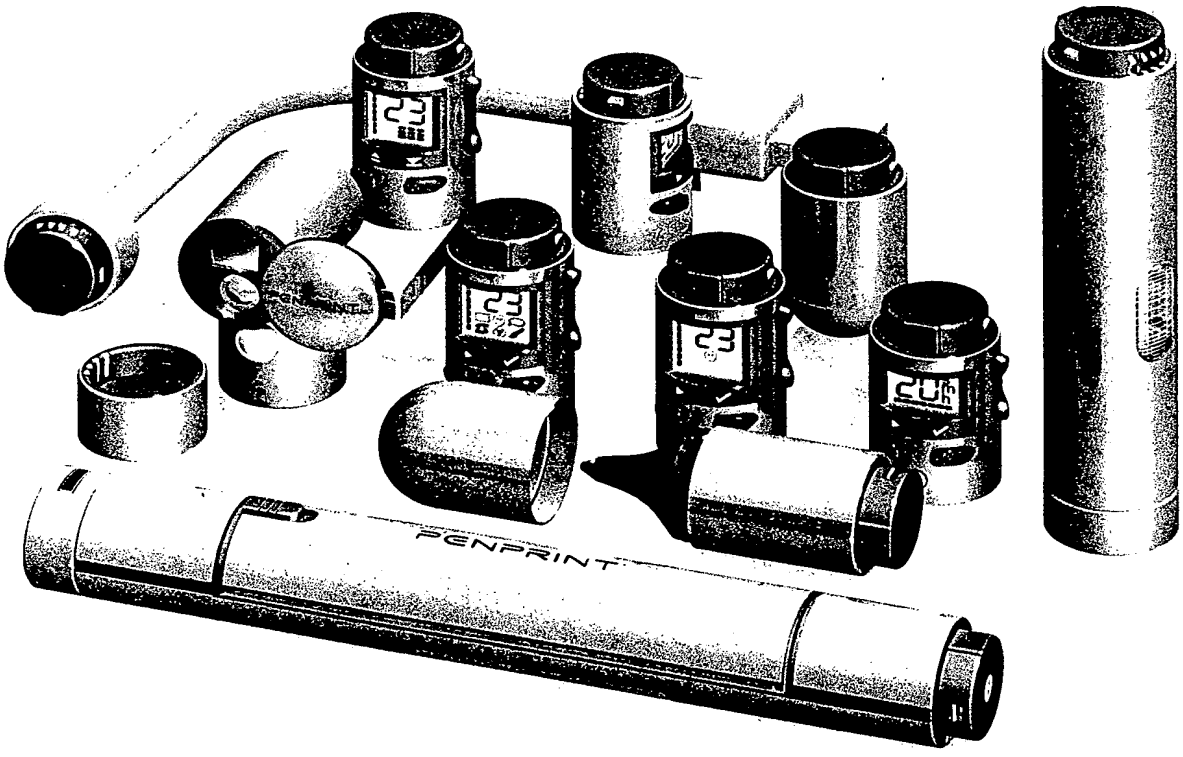
The Printer Module is capable of all methods of image access and therefore returns 1s for bits 0-3.

The 8-bit mode used for image read and write access via GetImage, StoreImage and TransferImage is interpreted as follows:

Table 3. 8-bit code used for read & write access

Bit	Interpretation
0	Orientation: 0 = 850 x 534, 1 = 534 x 850
1	0 = interleaved, 1 = planar
2-3	Color plane (valid only when bit 1 = planar) 00 = L* 01 = a* 10 = b* 11 = reserved
4-7	Reserved, and 0

PenPrint Module Overview



4 Introduction

PenPrint is designed to be a flexible, modular, extendable system. This section provides a summary of the various modules defined at this time for the PenPrint system:

- Printer Module
- Camera Module
- Memory Module
- USB Module
- Pen Module
- Flash module
- Special Effects module
- Character module
- Timer Module
- Gender changer module
- Laser pointer module

Life size (1:1) illustrations of the listed modules can be found in Figure 6, and example configurations produced by connecting various modules together can be found in Figure 7.

Additional possibilities for PenPrint modules (but not defined in this document) include:

- Pager module
- Infra-red module

The list of PenPrint modules is not exhaustive, as the PenPrint interface allows many more modules to be developed at some later date.

This section of the document is an *overview*. Later sections elaborate on the internals of specific modules as necessary. For example, "Printer Module" on page 50 elaborates on the design of the Printer Module, "Camera Module" on page 103 elaborates on the design of the Camera Module, and "Effects Module" on page 155 elaborates on the design of the Effects Module respectively.

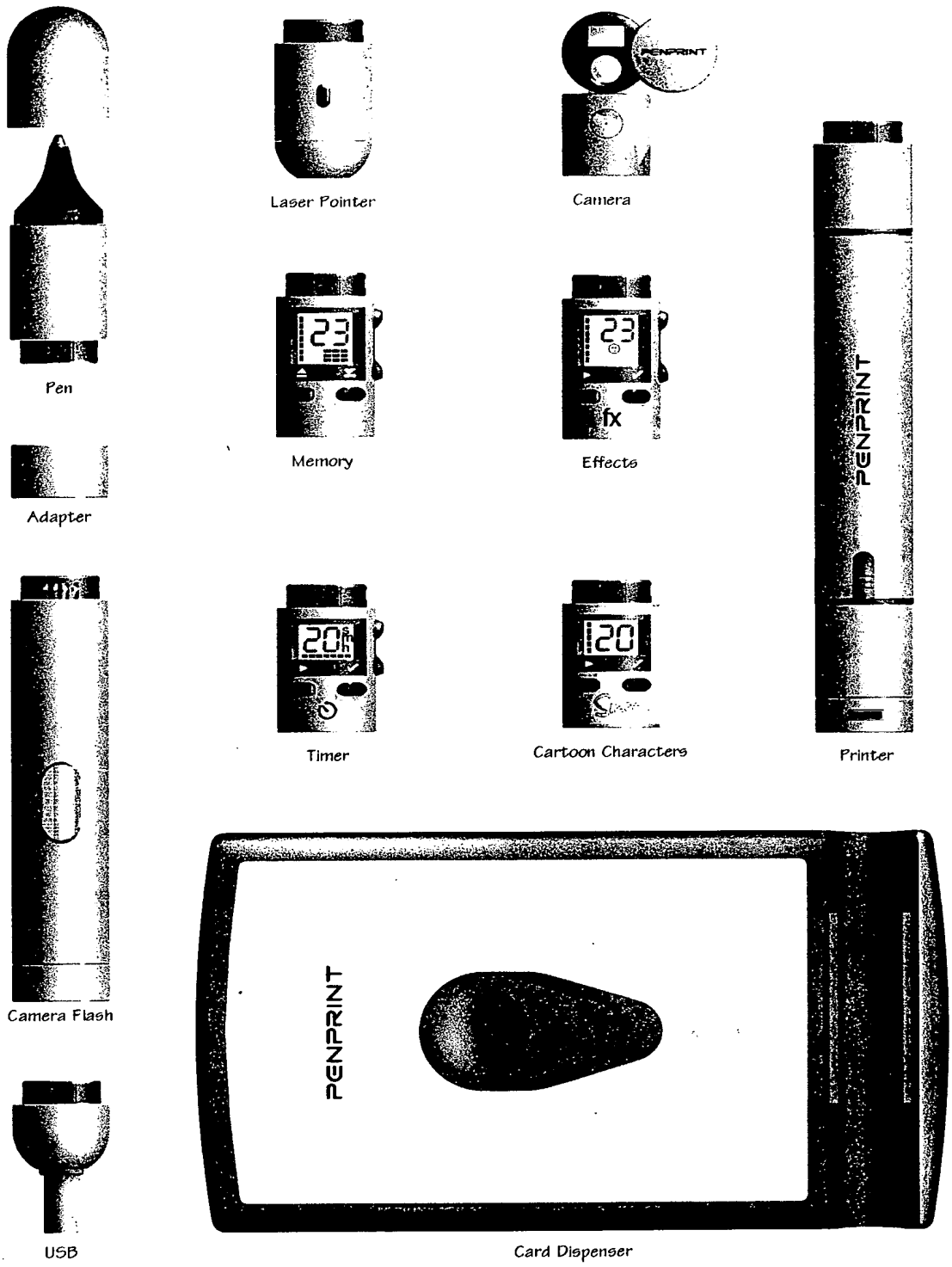


Figure 6. Life size (1:1) view of PenPrint modules

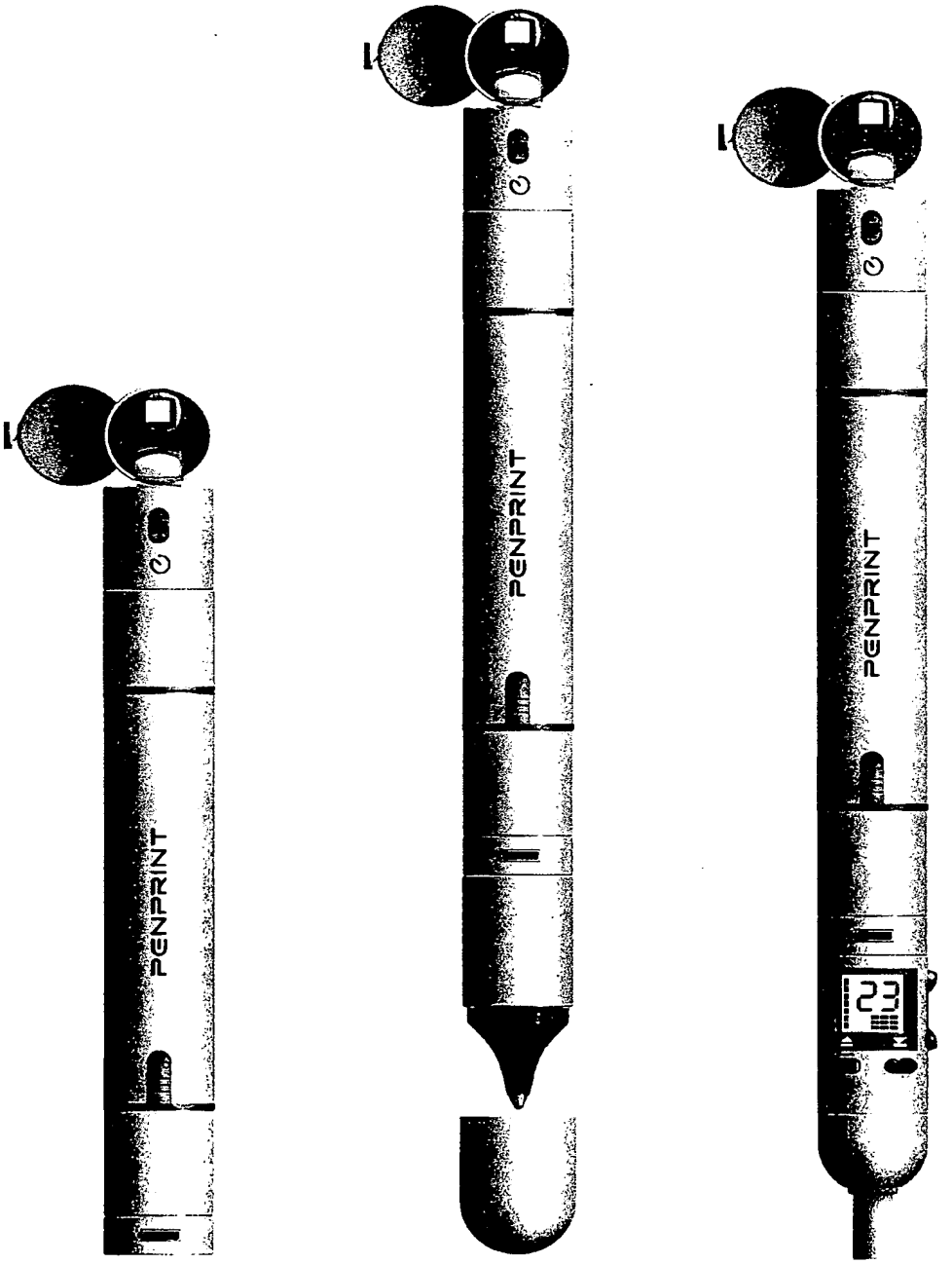


Figure 7. Example PenPrint Configurations (life size)

5 Printer Module

5.1 OVERVIEW

The PenPrint Printer Module is the central module in the PenPrint system. It contains a 2-inch Memjet printer, a Cyan/Magenta/Yellow ink cartridge, the current image stored in Flash memory, and a power source in the form of a 3V battery.

With regards to processing, the PenPrint Printer Module contains an image processing chip to print the stored image in high quality, and a QA chip [4, 5] for ensuring the ink cartridge does not run dry.

An optional card dispenser can be attached to the Printer Module to allow the easy dispensing of business cards into the printer.

5.2 APPEARANCE

Figure 8 shows the PenPrint Printer Module.

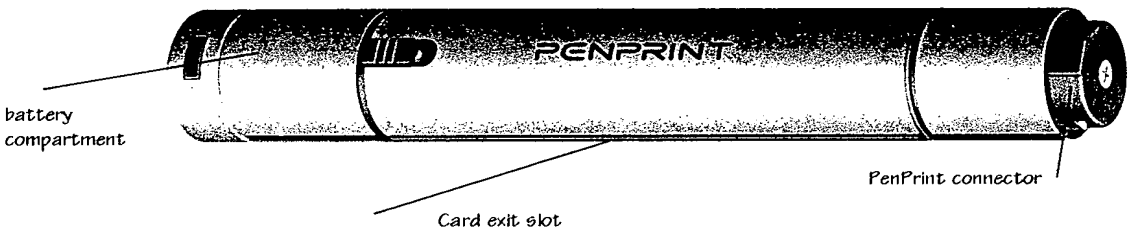


Figure 8. PenPrint Printer Module

5.3 METHOD OF OPERATION

The PenPrint Printer Module can be used as a stand-alone printer of a single image (such as business cards), or can be used in conjunction with other PenPrint modules to print a variety of images.

5.3.1 Printing Images

To print an image, a user simply inserts a business card into the input slot of the PenPrint Printer Module, as shown in Figure 9. Sensors detect the insertion and activate small motors to carry the card through the module. The printed card is ejected from the output

slot of the module over a time period of 1 second. There is no on/off switch - the act of inserting the card is the effective "on" switch for the duration of a print.

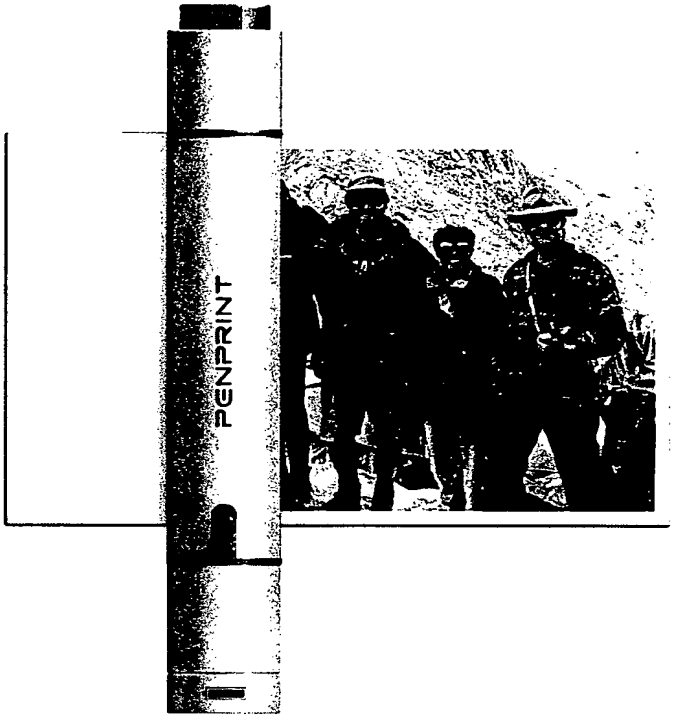


Figure 9. Printing an Image from a PenPrint Module (life size)

Multiple copies of a particular image can be obtained by the user inserting multiple blank cards, one at a time. A special card dispenser can also be used to print multiple cards. Figure 10 shows the optional card dispenser in isolation, and Figure 11 shows the dispenser as attached to the Printer Module.

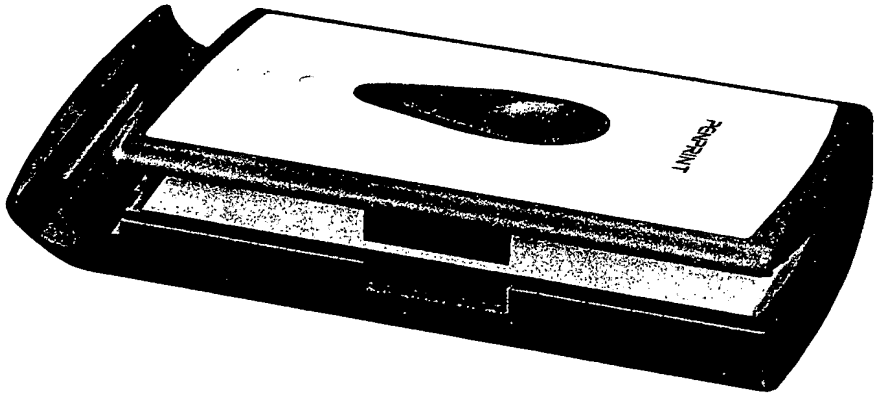


Figure 10. Optional Card Dispenser

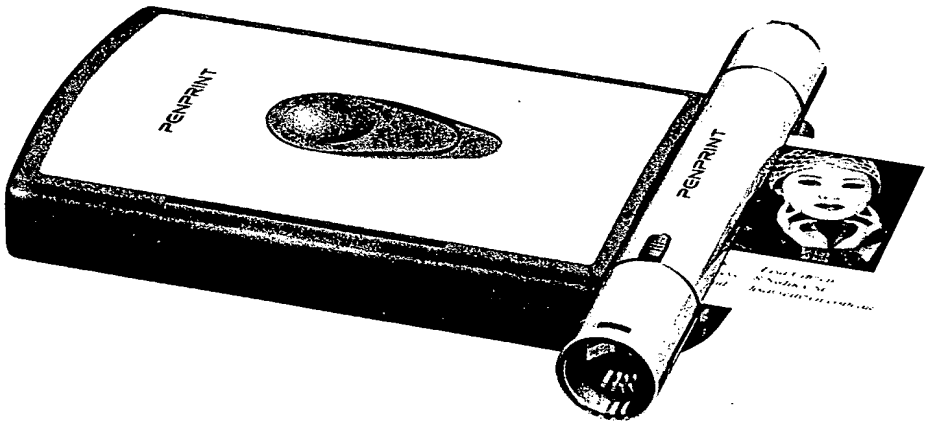


Figure 11. Printer Module with Attached Card Dispenser

The choice of which image is to be printed is achieved via additional modules. Images are sent from the specific module to the Printer module, or are copied from the Printer module to the specific module. The camera module can provide a photographic image, while the memory module provides previously captured images and computer processed images. See the other image processing modules for more information about the mode of operation.

5.3.2 Replacing ink

The volume of ink present in an ink cartridge is 450 µl (2mm × 3mm × 75mm), enough to produce 450 million dots of a given color. The exact number of images that can be printed before replacement will depend on the color composition of those images. 450 µl represents:

- 25 full black cards (black requires all three colors to be used)
- 50 full sized photos at 50% CMY coverage
- 111 typical photo/text cards at 22.5% CMY coverage
- 166 cards of black (CMY) text at 15% coverage

The embedded QA chip keeps track of how much ink has been used. If there is insufficient ink of any color to print a given image, the card will pass through the printer module, but nothing will be printed.

It is a simple matter to unclip the old ink cartridge and clip on a new one. Table 4 shows the breakdown of the 14 cent manufacturing cost for an ink cartridge.

Table 4. Breakdown of Manufacturing Cost for Ink Cartridge

Description	Cost ($)
2 Rollers @ 0.5c	0.01
Top molding	0.01
Bottom molding	0.01
450 µl Cyan ink @ $4/liter	0.0018
450 µl Magenta ink @ $4/liter	0.0018

Table 4. Breakdown of Manufacturing Cost for Ink Cartridge

Description	Cost ($)
450 µl Yellow ink @ $4/liter	0.0018
QA Chip	0.06
Label on inside	0.005
Elastomeric ink socket	0.01
Package	0.01
Assembly and Filling	0.02
TOTAL	0.14

5.3.3 Replacing battery

The battery used to power the PenPrint system is a CR1/3N cell. The battery contains enough power to print 133 photos. The characteristics of the battery are listed in Table 5:

Table 5. PenPrint Battery Characteristics

Parameter	Value
Type Designation	CR1/3N
Voltage (V)	3
Electrochemical System	Lithium
Typical Capacity (mAh)	170
Height (mm)	10.80
Diameter (mm)	11.60
Weight (g)	3.00

5.4 COMPUTER INTERFACE

The Printer Module holds a single image: the *current* image of the PenPrint system. Other PenPrint modules read the current image from the Printer Module, while others write a new current image to the Printer Module. Some modules have both read and write ability. The image transfer is accomplished by the PenPrint Serial Bus.

In addition, the USB Module makes all the PenPrint modules visible to an external computer system. This includes the Printer Module with its current image. A computer user can read the current image from the Printer Module or write a new image to the Printer Module for subsequent printing.

5.5 INTERNALS

The Printer Module consists of:

- standard PenPrint male/female bayonet connectors
- a 2-inch Memjet printer (see page 87)
- a replaceable ink cartridge
- an ASIC containing controlling logic and an image store (Flash memory)
- a QA chip [4, 5]
- a 3V power supply

Figure 12 shows an exploded view of the Printer Module, and Figure 13 shows the cross sectional view. A detailed discussion of the Printer Module can be found on page 50. A detailed discussion of the Memjet printer can be found on page 87.

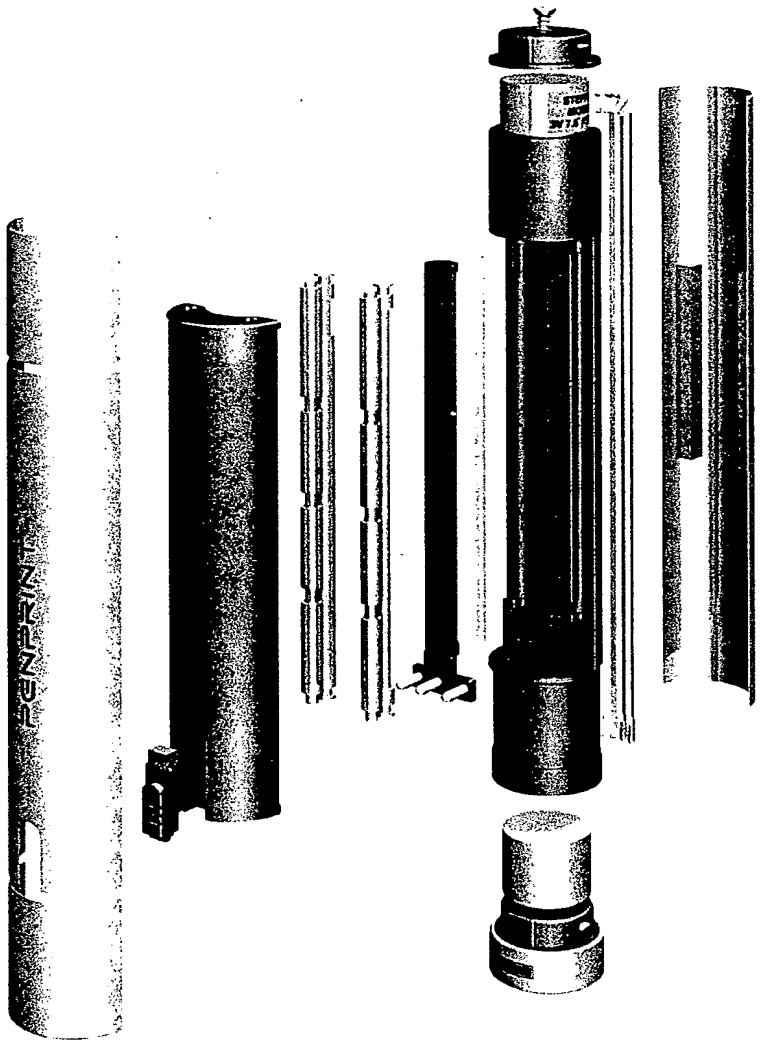
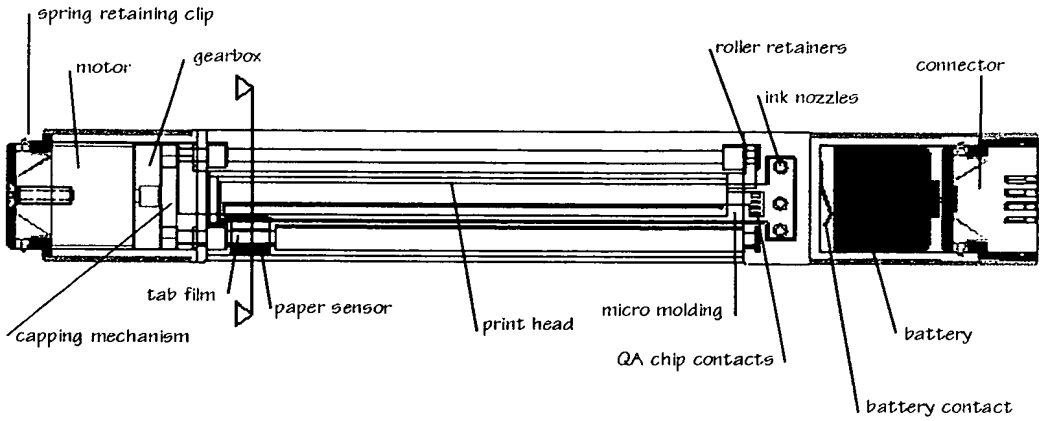


Figure 12. Exploded View of Printer Module



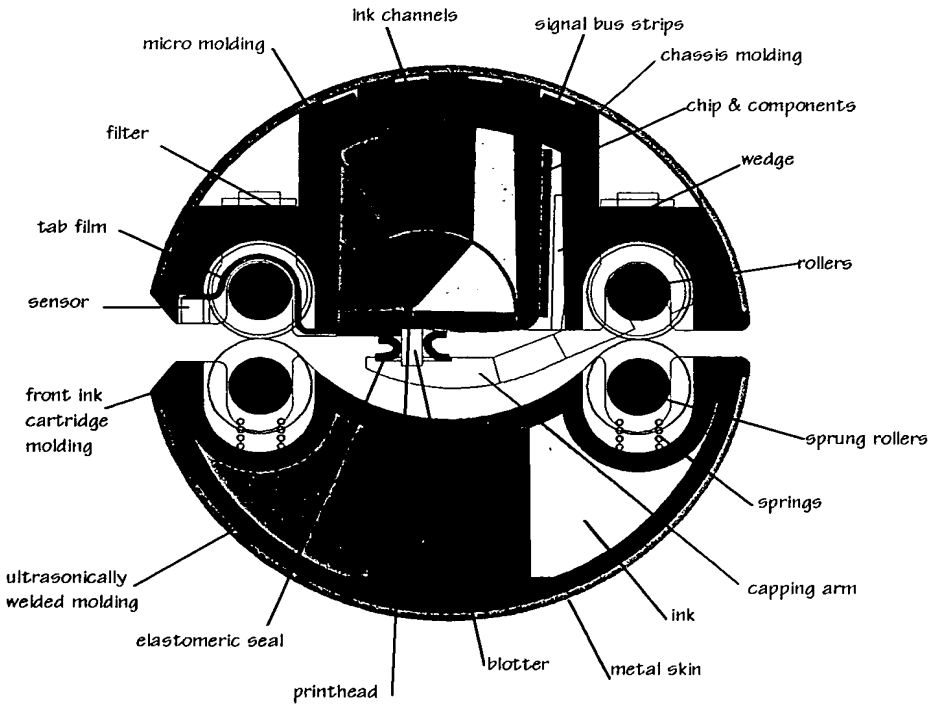


Figure 13. Cross Sectional View of Print Module

6 Camera Module

6.1 Overview

The PenPrint Camera Module provides a point-and-shoot camera component to a PenPrint system as a means of capturing images. The Camera Module is a standard PenPrint module containing an image sensor and specialized image processing chip. Captured images are transferred to the central PenPrint Printer Module for subsequent printing out, manipulation, or storage. The Camera Module also contains a self-timer mode similar to that found on regular cameras.

The Camera Module makes use of the Flash Module if it is detected in the current PenPrint configuration.

6.2 Appearance

Figure 14 shows a magnified illustration of the PenPrint Camera Module. Note the swivel connection for the lens assembly.

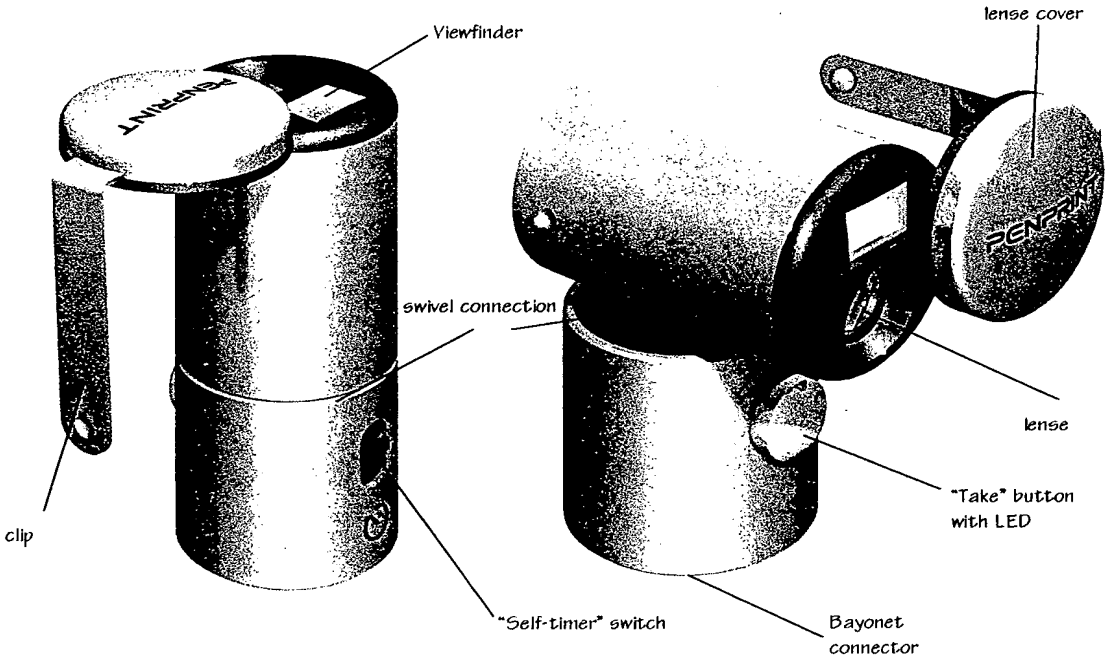


Figure 14. PenPrint Camera Module

6.3 Method of Operation

The Camera Module simply connects to a PenPrint configuration via the standard PenPrint bayonet connector. Power is provided from the PenPrint Printer module via the PenPrint Serial Bus.

To capture an image, a user simply presses the ***Take*** button. The viewfinder allows the user to frame the image before pressing the ***Take*** button. The swivel connection on the lens assembly assists in directing the lens to the correct orientation if the remainder of the PenPrint configuration is fixed.

When the ***Take*** button is pressed, the image is captured through the lens and transferred to the Printer Module. If the ***Take*** button is pressed again, a new image will be captured and transferred to the Printer Module. The image is always transferred to the Printer module once the Take button is pressed. Note that although the image remains in the Camera Module, there is no physical method of transferring the same image from the Camera Module again. The image must be saved from the Printer Module instead (to, for example, a Memory Module). The only way of directly accessing the captured image is via the computer interface (see Section 6.4 below). This limitation was chosen simply to reduce the usability complexity on the Camera Module.

The self-timer switch set to off/on disables and enables a 10 second delay between the pressing of the ***Take*** button and the capturing of the image. A LED inside the ***Take*** button provides a visual feedback during the countdown. The LED flashes once per second, and then stays on for the final two seconds of the countdown. The self-timing functionality is therefore identical to that of a conventional camera.

If there is an active Flash Module (see Section 9 on page 32) present in the PenPrint configuration, then the Flash will be activated depending on the Flash Module's flash mode. If the Flash Module has been turned ***off***, then the flash will not fire. If the Flash Module is set to ***auto***, then the flash fires as necessary (light detection carried out by the Camera Module). See Section 9 on page 32 for more information.

6.4 COMPUTER INTERFACE

The Camera Module can be instructed to take a photo either by a computer (via the USB Module) or by another module. However in both cases, the self-timer switch is ignored and the captured image is *not* transferred to the Printer Module. Instead, the image is simply captured and stored locally in the Camera Module. The Camera Module can then be instructed in a subsequent command to transfer its image to a specified module or simply to return it to the caller.

6.5 INTERNALS

The Camera Module consists of:

- standard PenPrint bayonet connector for easy attachment to a PenPrint configuration
- a simple lens assembly
- a CMOS imaging sensor
- a PenPrint Camera Image Processing chip (an ASIC designed to produce high quality images from the CMOS image sensor).

Partially and fully exploded views of the Camera Module internals are shown in Figure 15 and Figure 16. A detailed discussion of the Camera Module internals can be found on page 103.

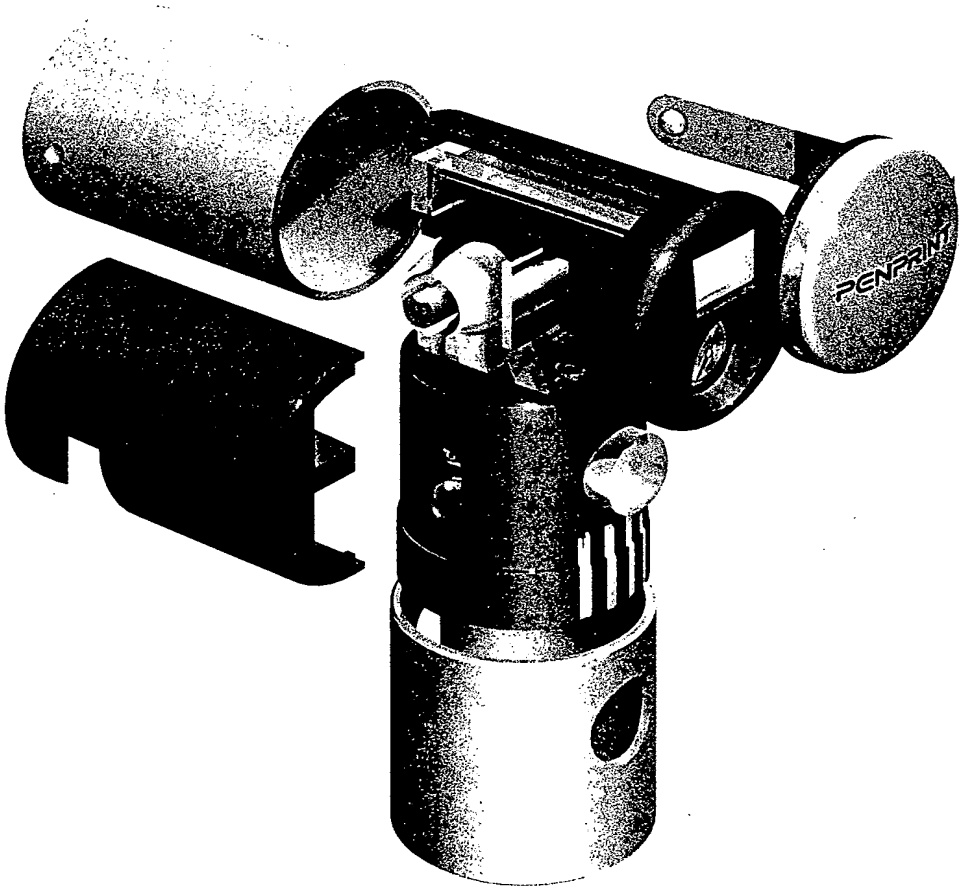


Figure 15. Partially Exploded View of Camera Module

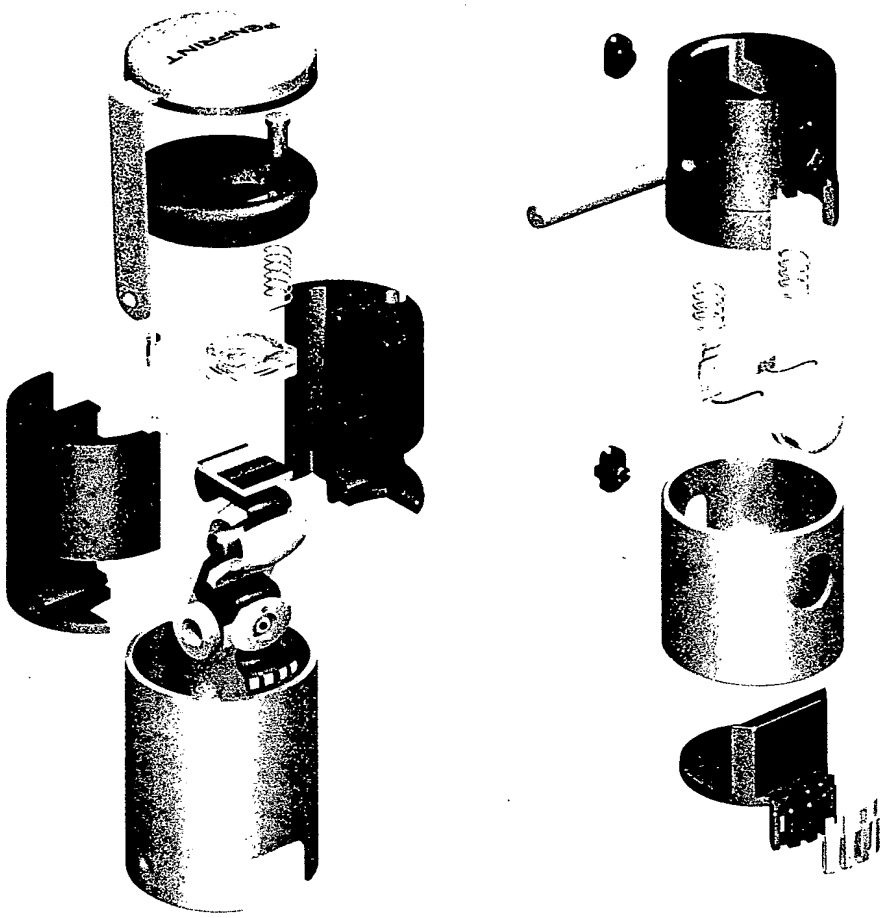


Figure 16. Fully Exploded View of Camera Module

7 Memory Module

7.1 Overview

The PenPrint Memory Module is a standard PenPrint module used for storing photographic images. A memory module stores 48 images, each of which can be accessed either at full resolution or at thumbnail resolution. Full resolution provides read and write access to individual images, and thumbnail resolution provides read access to 16 images at once in thumbnail form.

The Memory Module attaches to other PenPrint modules via the standard PenPrint bayonet connector. The male and female connectors allow the module to be connected at either end of a configuration. Power is provided from the PenPrint Printer Module via the PenPrint Serial Bus.

7.2 Appearance

Figure 17 shows the PenPrint Memory Module.

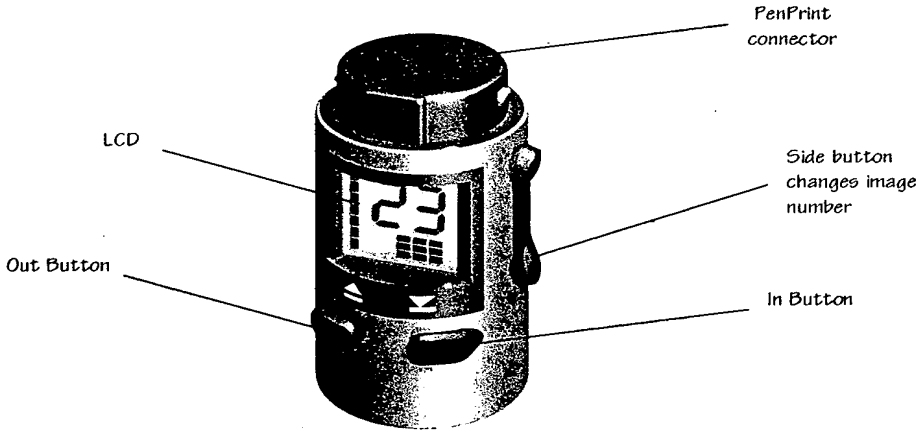


Figure 17. PenPrint Memory Module

The LCD provides visual feedback. It contains a 2-digit numerical display representing the image number. An icon below the number is visible when thumbnails are selected, and a series of block segments are also present to provide feedback during image transfer.

7.3 Method of Operation

The Memory Module is operated by 3 buttons and feedback is given via an LCD. The two principal operations are to transfer an image to the Printer Module, or to read an image from the Printer Module. In both cases, the LCD displays the number of the image read or sent.

The side up/down button changes the image number on the LCD display. The numbers simply cycle between 1 and the storage capacity of the module, followed by a number of thumbnail sets (1 to *n*, 1 for each 16). In a memory module that stores 48 images the numbers cycle between 1 and 48, followed by 1 to 3. While the thumbnail number is being shown, the thumbnail LCD icon is also displayed.

Pressing the ***Out*** button sends the current image number (or thumbnail set) to the Printer Module. Pressing the ***In*** button loads the image from the Printer Module into the named module. If the current setting is a thumbnail number the ***In*** button does nothing.

A small animation takes place during the transfer process to let the user know that the image transfer is taking place. The animation consists of a number of small black segments being enabled on the left side of the LCD showing the proportionate amount of data successfully transferred. The visual effect is that of a thermometer style status bar. The transfer time and animation time is approximately 1.5 seconds, but is important for user feedback.

7.4 COMPUTER INTERFACE

The Memory Module is also visible on the PenPrint Serial Bus as an image storage device. This enables the Memory Module to be controlled externally, either as a source or a destination for images. The USB Module makes any present Memory Modules visible to an external computer system, allowing the download or upload of images to each one.

When multiple Memory Modules are present in a PenPrint configuration, they are each controlled individually by the user. The operation of one Memory Module does not interfere with another.

Since each module can be controlled by a computer or by another module, it is possible to attach a PenPrint module that makes specific use of a Memory Module. One example of this kind of module is the PenPrint Timer Module. See Section 10 on page 34 for more information.

7.5 INTERNALS

The PenPrint Memory Module provides a black-box storage mechanism for PenPrint images. Any proprietary image storage format can be used, or changed over time since the stored image format is not externally available. Since memory storage for multiple images is still somewhat expensive, image compression is used. The PenPrint Memory Module therefore contains 2 chips:

- a 64 MBit (8 MB) Flash memory for image storage
- a controlling ASIC with on-board image compression hardware

The number of images that can be stored in the 64 MBit Flash memory is directly tied to the specific compression mechanism used.

7.5.1 Storage of Images

7.5.1.1 Requirement for Compression

Uncompressed, a PenPrint image requires 1.3 MB (850 × 534 × 3 bytes = 1,361,700 bytes). A 64 MBit DRAM would only be able to store 6 images. We therefore use JPEG compression to compress the contone data. Although JPEG is inherently lossy, for compression ratios of 10:1 or less the loss is usually negligible [8]. This yields a contone image size of 130KB, allowing the storage of between 50 and 60 images.

7.5.1.2 JPEG Compression

The JPEG compression algorithm [2] lossily compresses a contone image at a specified quality level. It introduces imperceptible image degradation at compression ratios below 5:1, and negligible image degradation at compression ratios below 10:1 [8].

JPEG typically first transforms the image into a color space which separates luminance and chrominance into separate color channels. This allows the chrominance channels to be subsampled without appreciable loss because of the human visual system's relatively greater sensitivity to luminance than chrominance. After this first step, each color channel is compressed separately.

The image is divided into 8×8 pixel blocks. Each block is then transformed into the frequency domain via a discrete cosine transform (DCT). This transformation has the effect of concentrating image energy in relatively lower-frequency coefficients, which allows higher-frequency coefficients to be more crudely quantized. This quantization is the principal source of compression in JPEG. Further compression is achieved by ordering coefficients by frequency to maximize the likelihood of adjacent zero coefficients, and then runlength-encoding runs of zeroes. Finally, the runlengths and non-zero frequency coefficients are entropy coded. Decompression is the inverse process of compression.

7.5.1.3 JPEG Compression in PenPrint

Images in the PenPrint system are in the L*a*b* color space, and are therefore already separated into luminance and chrominance channels. It is quite reasonable for the luminance channel to use one set of Huffman tables, and the chrominance channels to share another set. In addition, the chrominance channels can be subsampled. Finally, it should be noted that the images in PenPrint have a fixed resolution of 850 × 534. Any JPEG implementation can therefore be tailored to this resolution and does not have to be general.

At the end of compression, a JPEG bytestream is complete and self-contained. It contains all data required for decompression, including quantization and Huffman tables.

7.5.2 ASIC

A low complexity ASIC provides the processing power of the Memory Module. It contains:

- a JPEG core
- a PenPrint Serial Bus Interface core
- a microcontroller core
- a small amount of program ROM
- a small amount of RAM for program scratch

Images from the Memory Module must be made available in both planar and interleaved formats. How this is accomplished will depend on the ability of the JPEG core. If the core is only capable of compressing/decompressing a single channel at once, limited buffering may be required.

8 USB Module

8.1 OVERVIEW

The PenPrint USB Module is a module that allows the PenPrint system to be connected to a computer system. When so connected, images can be transferred between the computer and the various modules of the PenPrint system. The USB module allows captured images to be downloaded to the computer, and new images for printing to be uploaded into the PenPrint.

8.2 APPEARANCE

Figure 18 shows the appearance of the PenPrint USB Module. It is effectively a translator between standard USB cabling and the PenPrint serial bus cabling, with additional logic for translation of USB calls.

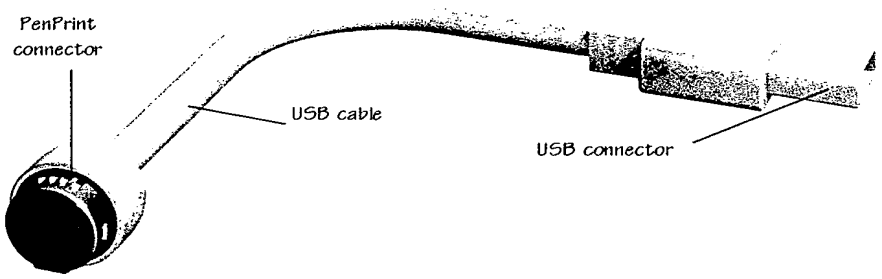


Figure 18. PenPrint USB Module

8.3 METHOD OF OPERATION

The PenPrint USB Module has a *physical* mode of operation and a *logical* mode of operation.

The physical mode of operation is where the user simply plugs the USB Module into a PenPrint system, and the USB connector into an appropriate USB connection on a computer system. There is no on/off switch on the PenPrint USB Module. The operating power is obtained from the computer's USB power supply rather than the PenPrint power supply.

Once physically connected, the logical mode of operation comes into play. The PenPrint USB Module translates the PenPrint modules into a virtual file system. Each PenPrint module appears as a named sub-directory, each containing a set of numbered image files. When a specific file is read, the USB Module translates the call into an image read command. When a specific file is written, the USB Module translates the call into an image write command.

In addition, each module appears as its own USB device, which allows the writing of drivers specific to each module. This is because the PenPrint USB Module acts as a standard USB hub.

9 Flash Module

9.1 OVERVIEW

The PenPrint Flash Module is used to generate a flash when taking photographs with the PenPrint Camera Module. The Flash module attaches to other PenPrint modules via the standard PenPrint bayonet connector, and contains its own power source.

The Flash Module is normally automatically selected by the Camera Module when required. However a simple switch allows the Flash Module to be explicitly turned off to maximize battery life.

9.2 APPEARANCE

Figure 19 shows the PenPrint Flash Module viewed from both sides..

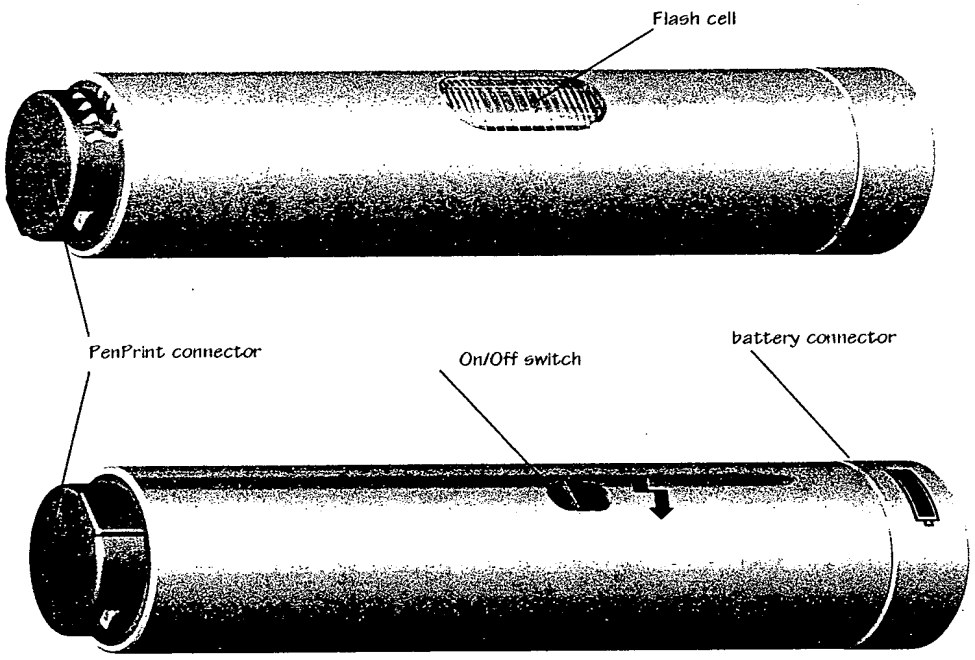


Figure 19. PenPrint Flash Module

9.3 METHOD OF OPERATION

The Flash Module is connected via the bayonet connectors just like any other PenPrint module, and functions very much like a flash unit that is attached to a regular camera. The Flash Module contains it own power supply to minimize impact on the rest of the PenPrint configuration.

To use the Flash Module, the user simply connects it to the PenPrint configuration and sets the switch to the appropriate mode. The single switch has two settings:

- **off** Whenever a photo is taken by the camera, the flash is not used. This is logically equivalent to not having the Flash Module connected.
- **auto** The Camera Module determines if a flash is required, and automatically generates a flash as required. The flash includes a pre-flash red-eye minimization cycle.

The Flash Module always works under control from the Camera Module. A flash is produced when a photo is taken. Appropriate charging of the flash takes place when the setting is on ***auto*** and the Camera Module determines that a flash is required. When the user presses the Take button on the Camera Module, the flash is discharged appropriately.

Theoretically any module or computer can instruct the Flash Module to charge/discharge etc., but it is expected that only the Camera Module will do so. Even the Timer Module, which instructs the Camera Module to take photos at a particular frequency does not interfere with the flash. The Flash Module remains under the control of the Camera Module. If Timer Module instructions occur too frequently for the Camera Module to process (in terms of flash requirements), the Camera Module must decide what to do. The most likely scenario is simply to take the photo as requested, but not to engage the flash if it has not been charged. For more information, see Section 6 on page 24 and Section 10 on page 34.

9.4 INTERNALS

The PenPrint Flash Module consists of:

- standard PenPrint male/female bayonet connectors for easy attachment to a PenPrint configuration
- a simple microcontroller and PenPrint Serial Bus Interface

State information is limited to:

- current setting (off, auto)
- current flash charge state (ready, not ready)
- pre-flash duration
- flash duration
- flash battery status

Instructions are limited to:

- charge
- pre-flash
- flash

A detailed discussion of the Camera Module internals can be found on page 103.

10 Timer Module

10.1 OVERVIEW

The PenPrint Timer Module is used to automate the taking of multiple photos with a PenPrint Camera Module, each photo separated by a specific time interval. The captured photos are stored in on-line Memory Module. Any flash requirements are handled by the Camera Module, and can therefore be ignored by the Timer Module.

The Timer Module takes its power from the Printer Module via the PenPrint serial bus.

10.2 APPEARANCE

Figure 20 shows the appearance of the Timer Module.

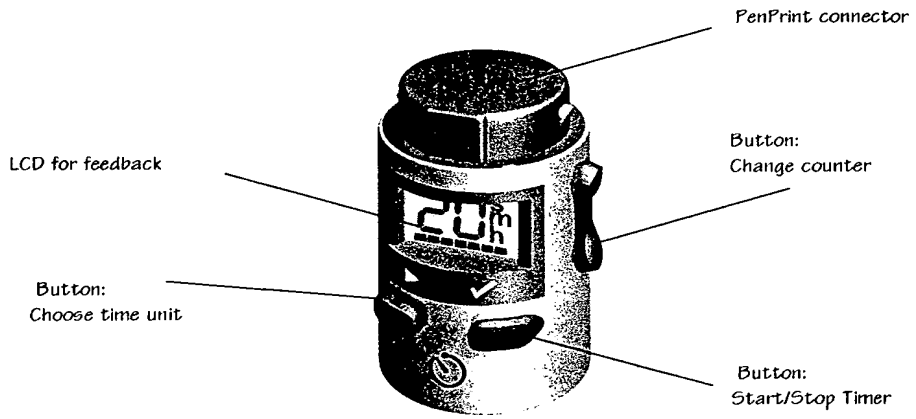


Figure 20. PenPrint Timer Module

The LCD provides visual feedback. It contains a 2-digit numerical display representing the number of time units between captured images. Three time units are available: seconds, minutes, and hours, represented by ***s***, ***m***, and ***h*** LCD segments respectively. A multi-segment thermometer provides an animated countdown between image captures.

10.3 METHOD OF OPERATION

The Timer Module must be used in conjunction with a Camera Module and at least one Memory Module. The first stage of Timer Module use is therefore to construct a PenPrint configuration containing at least these Modules. A Printer Module must also be present in order to provide power. The modules are joined together using the bayonet connectors.

Once told to start, the Timer Module counts down the specified time, and then instructs the Camera Module to capture and transfer an image to the appropriate Memory Module:

- **When there is only a single Memory Module**: Once told to start, the Timer Module continues to capture images every time interval and transfer them to the single Memory Module until that Memory Module is full.
- **When there are multiple Memory Units**: The process is the same as for the single Memory Module, except that the Memory Modules are filled up one by one until all Memory Modules have been filled. The first Memory Module to be written to is the

one physically *closest* to the Camera Module, and the last Memory Module to be written to is the one physically *furthest* from the Camera Module.

Two tasks need to be accomplished before the Timer Module can be activated to begin the countdown. The tasks can be performed in any order.

- **Set the delay between the capture of each photo.** This is accomplished by selecting the time unit using the front button, and then selecting the number of those units by the side button. For example, a time delay of 30 seconds can be accomplished by setting the unit to seconds, and then adjusting the number to 30. A time delay of 15 minutes can be accomplished by setting the unit to minutes and then adjusting the number to 15. The Timer Unit maintains a context for each unit so as to minimize the change required by the user.
- **Set the initial photo number on the Memory Modules.** The captured images will be written to the Memory Modules present in the configuration, starting from the module closest to the Camera Module, and ending at the module furthest from the Camera Module. The first image to be written to a specific module will be written to the current image number on that module. The image number is then incremented. Setting the image number before the Timer is activated lets the user specify how many images each Memory Module will capture. For example, setting a 48 image Memory Module to 40 allows the capture of 9 images: images 40, 41, 42, 43, 44, 45, 46, 47, and 48. Once the image count reaches 48 it stops there and no further images are written to that Memory Module.

Once these two tasks have been done, the Timer Module's ***Activate*** button can be pressed. Pressing the ***Activate*** button a second time deactivates the Timer Module. While the Timer Module is actively counting down, feedback is given in two ways. Firstly, the time unit LCD segment flashes on and off (1 second on, 1 second off) to let a user know the countdown is active. Secondly, a multi-segment thermometer provides an animated countdown of the proportion of time elapsed until the next image capture time.

10.4 INTERNALS

The PenPrint Timer Module contains a simple microcontroller and PenPrint Serial Bus Interface. State information is limited to:

- current time unit setting (seconds, minutes, hours)
- number of time units (2-60 seconds, 1-60 minutes, 1-96 hours)
- timer active (yes, no)
- time until next image capture (2-60, 1-60, 1-96)
- current Camera target
- current Memory Module target
- next Memory Module image number

Instructions are limited to:

- set time unit
- set time amount
- activate timer
- deactivate timer

11 Laser Pointer Module

11.1 Overview

The PenPrint Laser Pointer Module is a laser pointer embedded in a PenPrint module. The Laser Pointer Module is an isolated module, in that it does not perform any image capture, storage, or processing.

Like the PenPrint Pen Module, the PenPrint Laser Pointer Module exists as a functional addition to the central PenPrint system. It is provided because laser pointer services are typically incorporated into other pen-like devices.

The PenPrint Laser Pointer Module contains its own power supply, so does not use the PenPrint system power. The Laser Pointer Module does not appear as a device on the PenPrint Serial Bus, and does not process any image data.

11.2 Appearance

Figure 21 shows the PenPrint Laser Pointer Module. Note the single button for operation, and the connector for attachment to other PenPrint modules.

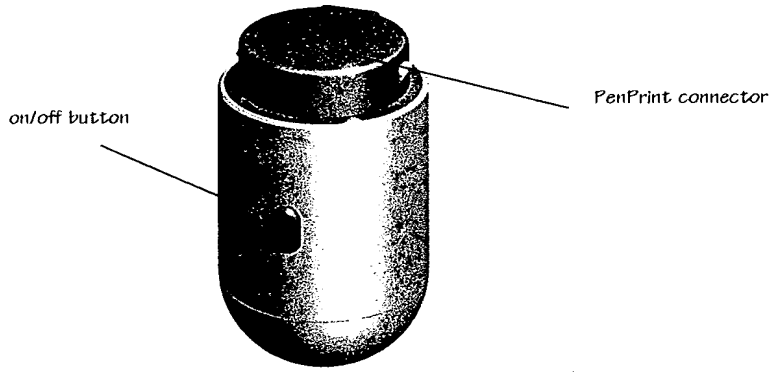


Figure 21. PenPrint Laser Pointer Module

11.3 Method of Operation

A single button on the side of the module enables the laser pointer. While the button is pressed down, the laser pointer operates. When the button is released, the laser pointer stops operating. The Laser Pointer Module functions completely independently of a PenPrint configuration.

11.4 Internals

The Laser Pointer Module contains a battery, an on/off button, and laser pointer unit. The battery is connected to the laser pointer unit via the on/off button. There are no ASICs or processors of any kind.

12 Effects Module

12.1 OVERVIEW

The PenPrint Effects Module is an image processing module. It allows a user to select a number of effects and applies them to the current image in the PenPrint Printer Module. The effects include borders, clip-art, captions, warps, color changes, and painting styles.

The PenPrint Effects Module obtains power from the PenPrint Serial Bus.

12.2 APPEARANCE

Figure 22 shows the PenPrint Effects Module. Note that the LCD panel is showing all segments active.

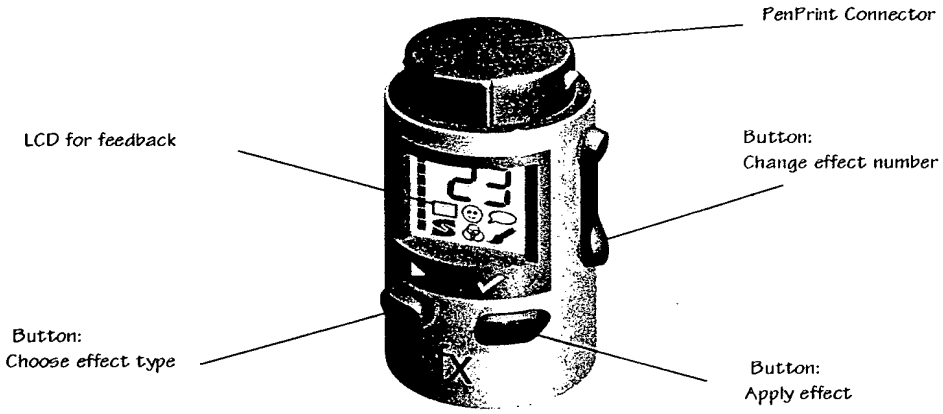


Figure 22. PenPrint Effects Module Showing all LCD Segments

12.3 METHOD OF OPERATION

The basic level of operation is to select an effect and then to apply that effect to the current image in the PenPrint Printer Module.

The LCD provides feedback in the form of an effect type and number, and an animated thermometer that is shown while the effect is being applied to the image.

Pressing the Effect Type button cycles through the effect types. Each effect type has its own LCD icon and a current effect number. The effect types are borders, clip-art, captions, warps, color changes, and painting styles.

Pressing the side up/down button increments or decrements the effect number. The effect number is visible on the LCD screen. Ideally each effect type has 100 effects (although this is not strictly required). The current effect number for each effect type is remembered by the Effect Module so that the next time the effect type is chosen, the previously used effect number is selected.

Once the desired effect type and effect number have been chosen, the effect can be applied to the image currently in the PenPrint Printer Module. This is done by pressing the Apply

button below the LCD. As the image is being read, the effect applied, and resultant image written back, a series of animated segments appears on the left side of the LCD to form a thermometer-style status bar. The proportion of segments displayed is the proportion of the work that has been done so far. The user knows that the effect has been applied once all the segments have been displayed.

The user must consult a manual to determine the correspondence between effect and effect number.

To apply multiple effects to an image a user simply applies each effect one at a time. The order may be important if a specific end image is desired. For example, adding a border and then warping the image will produce a different result to warping the image followed by adding a border.

12.4 EFFECT TYPES

The Effects module provides 6 basic effect types:

- borders
- clip-art
- captions
- warps
- color changes
- painting styles

These effects can be combined in any way and in any order. Figure 23 shows two life size samples of printouts from the Effects Module.

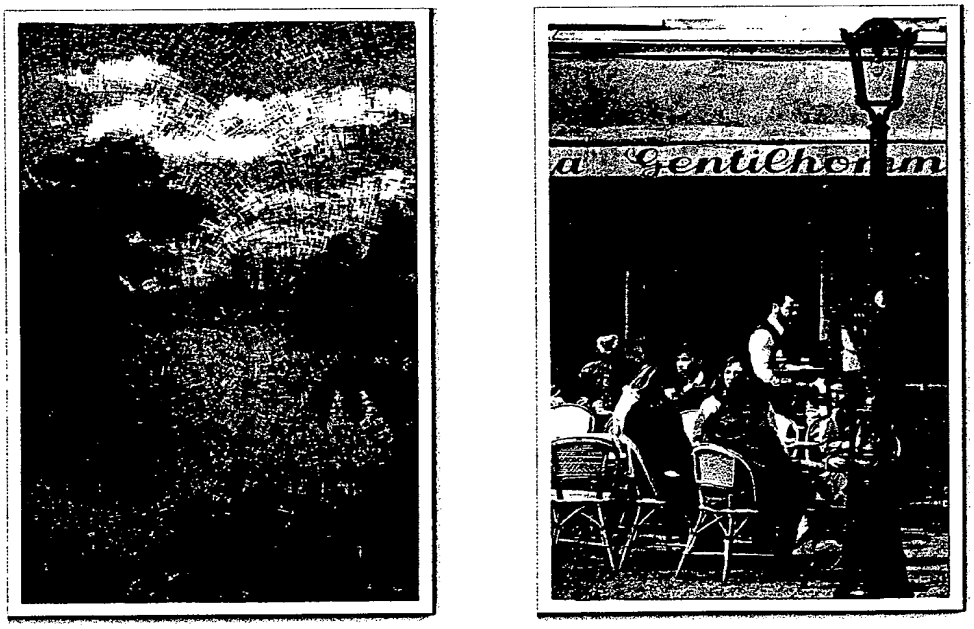


Figure 23. Sample Images with Effects Applied (life size)

12.4.1 Borders

Image borders can range from simple white to elaborate and complex designs. Some borders incorporate the image, but the majority of borders and frames are straightforward, consisting of one image being overlaid over another. Figure 24 shows some life sized examples of border effects.

Figure 24. Sample Border Effects (life size)

12.4.2 Clip-art

Clip-art consist of photographic and stylized cartoon characters and images inserted into the image at a specific location. The characters and clip-art included in a general Effects Module would require little or no licensing (compared to the Kid's Character Module described below). Examples include clowns, action figures, smiley faces, ugly faces, animals, specific objects etc.

Note that no effort is made to interpret the image before the character is added. For a given effect number a specific character is added at a specific fixed place in the image.

The Character Module, by contrast, only contains Clip-art effects of a given topic or genre. Examples include The Simpsons, Star Wars, Batman, and Dilbert as well as company specific modules for McDonalds etc. See "Character Module" on page 46 for more information. Figure 25 shows some life sized examples of clip-art effects.

Figure 25. Sample Clip-art Effects (life size)

12.4.3 Captions

Captions consist of speech bubbles, lines of text or other textual effects applied to an image. Examples include "Happy Birthday!", "How old are you?", "It was *this* big!", "Wow!", "Help!" etc.

Note that no effort is made to interpret the image before the caption is added. For a given effect number a specific caption is added at a specific fixed place in the image.

Captions are overlaid as graphics. Therefore they can include anything and be in any language, any color combination, and be in multiple fonts. The user is unable to change any part of the caption or style - the caption is simply added to the image. Choice is available due to the 100 caption numbers. Figure 26 shows some life sized examples of caption effects.

Figure 26. Sample Caption Effects (life size)

12.4.4 Warps

Images can be warped using the Warp effect type. Each warp effect number affects a fixed area of the image, which can be the entire image, or a limited part of the image. As with other effect modes, no effort is made to interpret the image before the warp is applied. For a given effect number, the specific warp is added at a specific fixed place in the image.

Some warps are specifically designed to affect a face image, although there is no specific face detection mechanism. Instead, the warp is applied to the part of the image where the face *should* be. Of course there may not be a face there at all, so the warp will only be applied to that fixed area of the image. Figure 27 shows some life sized examples of warp effects.

Figure 27. Sample Warp Effects (life size)

12.4.5 Color Changes

It is often desirable to transform an image in terms of color. Simple color effects include removal of color to produce a greyscale image or a sepia tone image. More complex effects include exaggeration of certain colors, substitution of one color for another etc. Figure 28 shows some life sized examples of color effects.

Figure 28. Sample Color Effects (life size)

12.4.6 Painting Styles

The painting styles effect mode lets the user apply a painting style to an image. For example, an image can be changed into an impressionist drawing of the original, or drawn with a variety of brush strokes from a limited color palette. Painting styles also includes limited tiling effects.

It should be noted that these painting styles do not utilize lighting calculations. Certain types of metallic or bump-map based painting effects are therefore not possible. Figure 29 shows some life sized examples of painting effects.

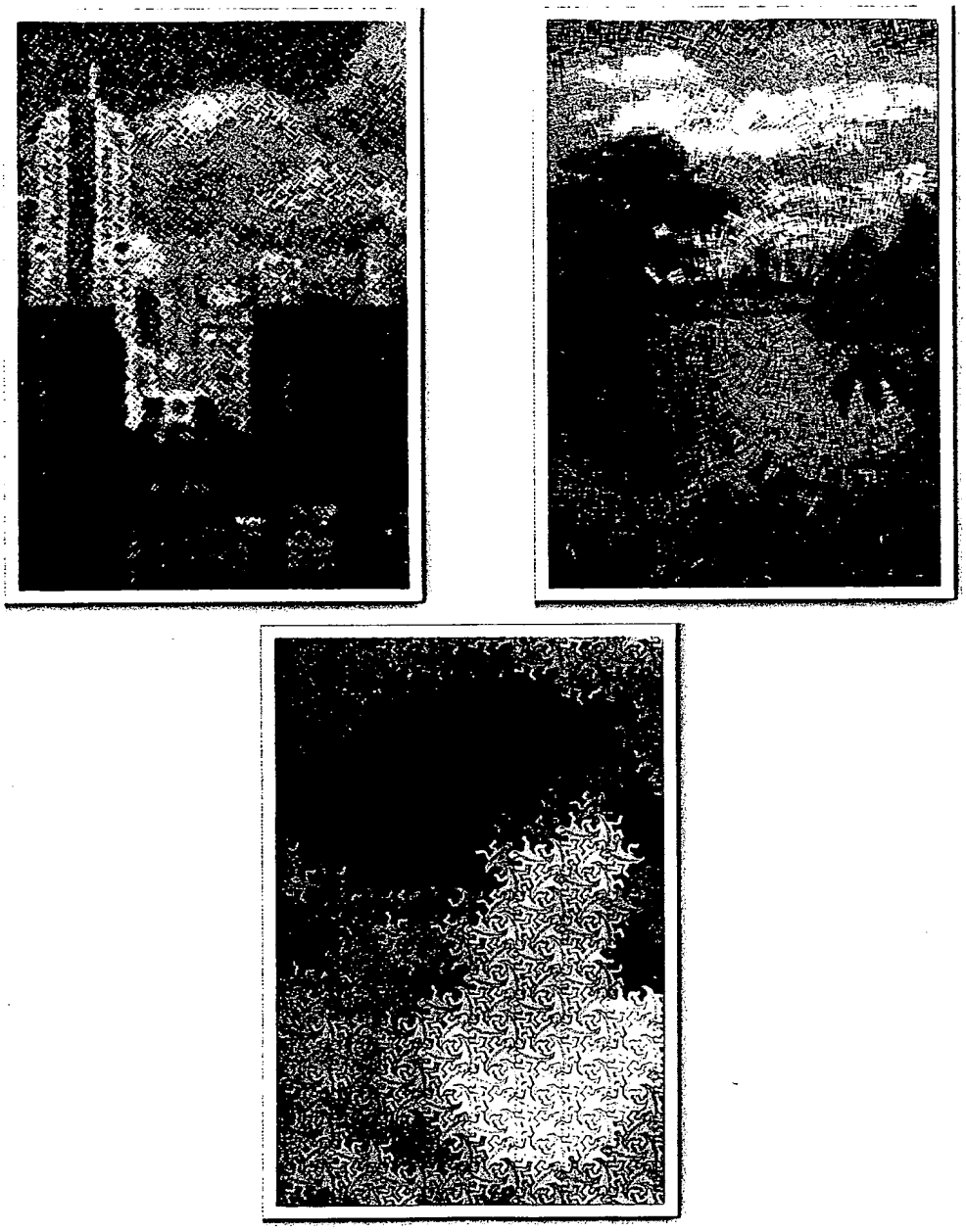


Figure 29. Sample Painting Effects (life size)

12.5 INTERNALS

The Effects Module is built with:

- a standard PenPrint male bayonet connector
- a standard PenPrint male bayonet connector
- a special-purpose LCD
- 2 single buttons and 1 double button
- an Effects Module Central Processor (an ASIC containing a CPU, DSP, and an implementation of the VARK image processing language [7] for producing the effects)
- a ROM for program and data

An exploded view of the Effects Module can be found in Figure 30. A detailed discussion of the Effects Module internals can be found on page 155.

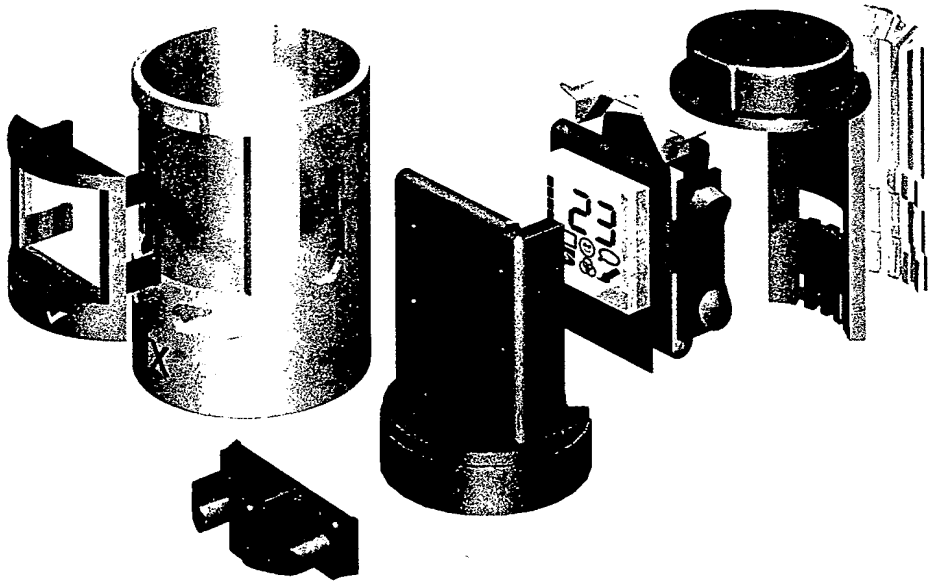


Figure 30. Exploded View of Effects Module

13 Character Module

13.1 OVERVIEW

The PenPrint Character Module is a special type of PenPrint Effects Module (see page 37) that only contains character clip-art effects of a given topic or genre. Examples include The Simpsons, Star Wars, Batman, and Dilbert as well as company specific modules for McDonalds etc. As such it is an image processing module.

The PenPrint Character Module obtains power from the PenPrint Serial Bus.

13.2 APPEARANCE

Figure 31 shows a sample PenPrint Effects Module for The Simpsons.

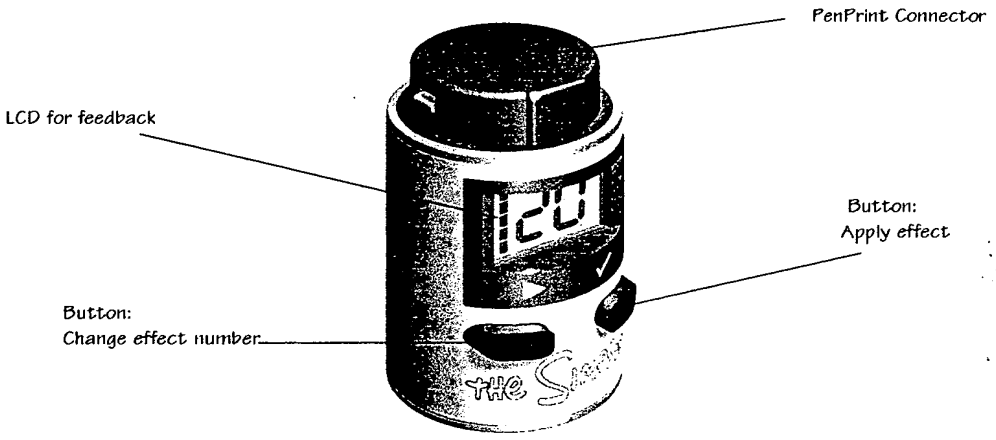


Figure 31. PenPrint Character Module for *The Simpsons*

13.3 LICENSING ISSUES

The majority of Character Modules are likely to be based on licensed characters from other companies. Movie and television studios such as Disney, Dreamworks, Paramount, Warner Bros., Fox, and others can produce movie or TV tie-in Character Modules. Companies such as McDonalds can produce product/personality based Character Modules, and other manufacturers can produce simple theme based Character Modules, such as Halloween, Christmas, Pirates, Butterflies etc. Limited editions and special collector's editions can also be produced.

13.4 TECHNICAL ISSUES

The PenPrint Character Module is essentially a special purpose Effects Module. Since the characters effects mode is built into the Effects Module, the Character Module is a limited subset of functionality from the Effects Module.

The major *physical* difference between the Character Module and the Effects Module is that on the Character Module there is no side button. The Character Module also uses a smaller ROM since there is only character overlay. No complex painting effects or warps

are required, so strictly speaking there is no need for the Vark interpreter to be included. However it may be easier to simply leave the internals the same as the Effects Module, and thereby allow the same subset of Vark for the effects. The ROM size is determined by the number of character effects and the size of the graphics. The alternative is to allow only Vark compositing. The actual decision can be based on levels of sales, complexity of imaging, and retail price expectations.

13.5 METHOD OF OPERATION

The basic level of operation is to select an effect and then to apply that effect to the current image in the PenPrint Printer Module.

The LCD provides feedback in the form of an effect type and number, and an animated thermometer that is shown while the effect is being applied to the image.

Pressing the Effect Type button cycles through the effect types. Each effect type has its own LCD icon and a current effect number. The effect types are borders, characters, captions, warps, color changes, and painting styles.

Pressing the side up/down button increments or decrements the effect number. The effect number is visible on the LCD screen. Ideally each effect type has 100 effects (although this is not strictly required). The current effect number for each effect type is remembered by the Effect Module so that the next time the effect type is chosen, the previously used effect number is selected.

Once the desired effect type and effect number have been chosen, the effect can be applied to the image currently in the PenPrint Printer Module. This is done by pressing the Apply button below the LCD. As the image is being read, the effect applied, and resultant image written back, a series of animated segments appears on the left side of the LCD to form a thermometer-style status bar. The proportion of segments displayed is the proportion of the work that has been done so far. The user knows that the effect has been applied once all the segments have been displayed.

The user must consult a manual to determine the correspondence between effect and effect number.

To apply multiple effects to an image a user simply applies each effect one at a time. The order may be important if a specific end image is desired. For example, adding a border and then warping the image will produce a different result to warping the image followed by adding a border.

14 Gender Changer Module

14.1 OVERVIEW

The PenPrint Gender Changer Module is a female/female connector that allows male connector provides a housekeeping service of allowing a chain order reversal allows PenPrint modules to be attached to the laser pointer embedded in a PenPrint module. The Gender Changer Module is a housekeeping module, in that it facilitates the use of other modules, and does not perform any specific processing of its own.

All "through" PenPrint modules have a male connector at one end, and a female connector at the other end. The modules can therefore be chained together, with each module connected at *either* end of the PenPrint chain. However some modules, such as the Pen Module, are terminating modules, and therefore have either a male or female connector only. Such single-connector modules can only be connected at one end of the PenPrint chain. If two such modules are to be connected at the one time, a Gender Changer Module is required.

14.2 APPEARANCE

Figure 32 shows the PenPrint Gender Changer Module. Note that both connectors are female.

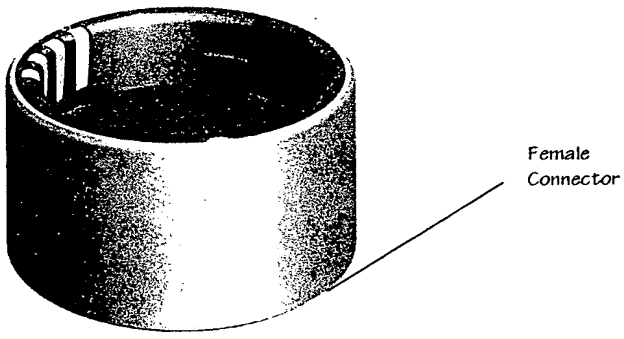


Figure 32. PenPrint Gender Changer Module

14.3 METHOD OF OPERATION

As the Gender Changer Module has no logical function, there is no logical user interface. The Gender Changer Module is purely to facilitate connection of male PenPrint modules to the male end of a PenPrint configuration. The physical user interface is therefore purely at construction time of a PenPrint configuration, and not during the use or operation of that configuration.

15 Pen Module

15.1 OVERVIEW

The PenPrint Pen Module is a pen in a PenPrint Module form. It is an isolated module in that it attaches to a PenPrint system but is completely independent of any other module. It does not consume or require any power. The PenPrint Pen Module is defined because it is expected of a pen shaped, pen sized device, especially with a name like PenPrint.

15.2 APPEARANCE

Figure 33 shows the PenPrint Pen Module.

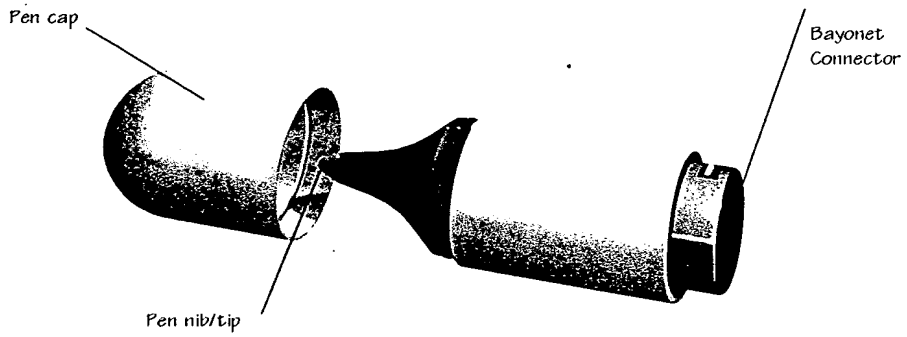
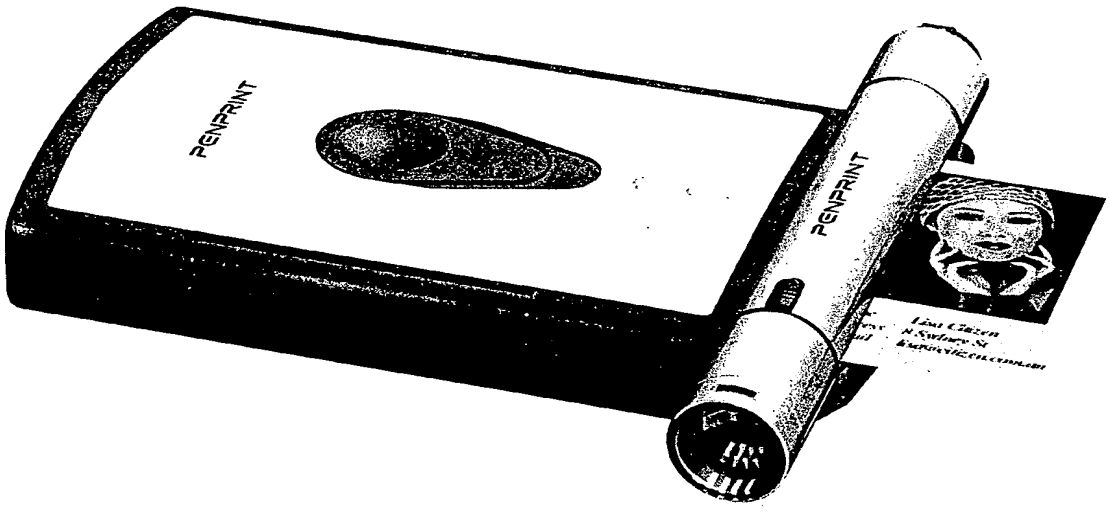


Figure 33. PenPrint Pen Module

15.3 METHOD OF OPERATION

The PenPrint Pen Module simply behaves as a pen. The cover can be removed and the pen used for writing. The nib end can be unscrewed from the main body of the module and the ink store replaced. It is envisaged that ink will be available in a variety of colors, including black, blue, and red.

Printer Module



16 Introduction

16.1 OVERVIEW

The PenPrint Printer Module is the central module in the PenPrint system. It contains a 2-inch Memjet printer (see page 87), a Cyan/Magenta/Yellow ink cartridge, the current image stored in Flash memory, and a power source in the form of a 3V battery.

With regards to processing, the PenPrint Printer Module contains an image processing chip to print the stored image in high quality, and a QA chip [4, 5] for ensuring the ink cartridge does not run dry.

An optional card dispenser can be attached to the Printer Module to allow the easy dispensing of business cards into the printer.

16.2 APPEARANCE

Figure 34 shows the PenPrint Printer Module.

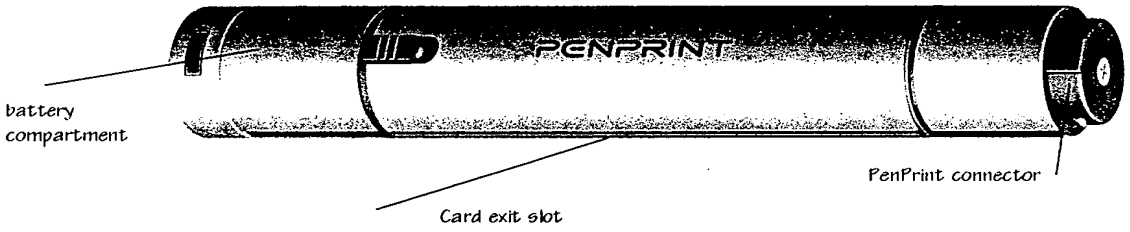


Figure 34. PenPrint Printer Module

16.3 METHOD OF OPERATION

The PenPrint Printer Module can be used as a stand-alone printer of a single image (such as business cards), or can be used in conjunction with other PenPrint modules to print a variety of images.

16.3.1 Printing Images

To print an image, a user simply inserts a business card into the input slot of the PenPrint Printer Module, as shown in Figure 35. Sensors detect the insertion and activate small motors to carry the card through the module. The printed card is ejected from the output

slot of the module over a time period of 1 second. There is no on/off switch - the act of inserting the card is the effective "on" switch for the duration of a print.

Figure 35. Printing an Image from a standalone Printer Module (lifesize)

Multiple copies of a particular image can be obtained by the user inserting multiple blank cards, one at a time. A special card dispenser can also be used to print multiple cards. Figure 36 shows the optional card dispenser in isolation, and Figure 37 shows the dispenser as attached to the Printer Module.

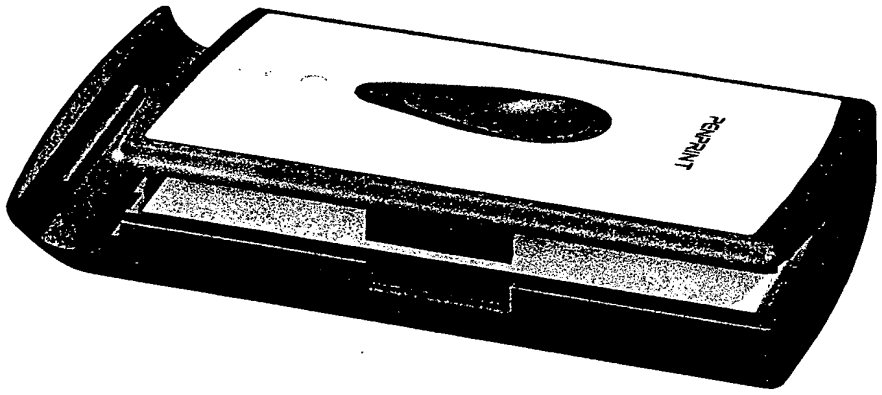


Figure 36. Optional Card Dispenser

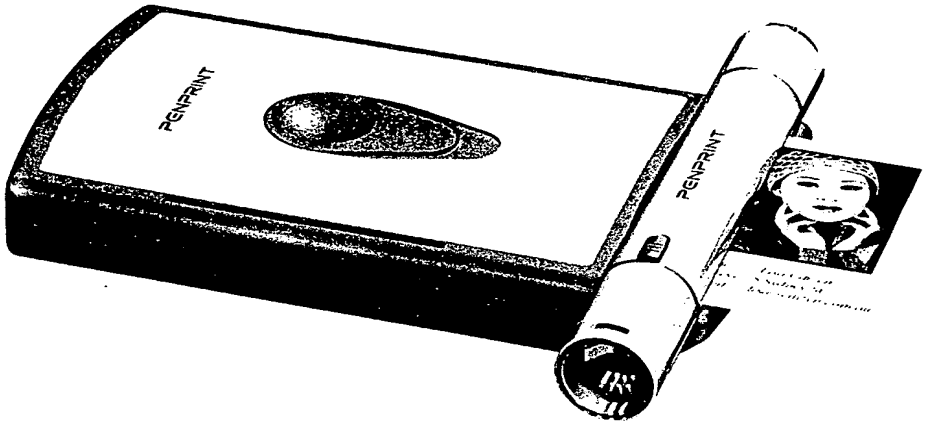


Figure 37. Printer Module with Attached Card Dispenser

The choice of which image is to be printed is achieved via additional modules. Images are sent from the specific module to the Printer module, or are copied from the Printer module to the specific module. The camera module can provide a photographic image, while the memory module provides previously captured images and computer processed images. See the other image processing modules for more information about the mode of operation.

16.3.2 Replacing ink

The volume of ink present in an ink cartridge is 450 µl (2mm × 3mm × 75mm), enough to produce 450 million dots of a given color. The exact number of images that can be printed before replacement will depend on the color composition of those images. 450 µl represents:

- 25 full black cards (black requires all three colors to be used)
- 50 full sized photos at 50% CMY coverage
- 111 typical photo/text cards at 22.5% CMY coverage
- 166 cards of black (CMY) text at 15% coverage

The embedded QA chip keeps track of how much ink has been used. If there is insufficient ink of any color to print a given image, the card will pass through the printer module, but nothing will be printed.

It is a simple matter to unclip the old ink cartridge and clip on a new one. Table 6 shows the breakdown of the 14 cent manufacturing cost for an ink cartridge.

Table 6. Breakdown of Manufacturing Cost for Ink Cartridge

Description	Cost ($)
2 Rollers @ 0.5c	0.01
Top molding	0.01
Bottom molding	0.01
450 µl Cyan ink @ $4/liter	0.0018
450 µl Magenta ink @ $4/liter	0.0018

Table 6. Breakdown of Manufacturing Cost for Ink Cartridge

Description	Cost ($)
450 µl Yellow ink @ $4/liter	0.0018
QA Chip	0.06
Label on inside	0.005
Elastomeric ink socket	0.01
Package	0.01
Assembly and Filling	0.02
TOTAL	0.14

16.3.3 Replacing battery

The battery used to power the PenPrint system is a CR1/3N cell. The battery contains enough power to print 133 photos. The characteristics of the battery are listed in Table 7:

Table 7. PenPrint Battery Characteristics

Parameter	Value
Type Designation	CR1/3N
Voltage (V)	3
Electrochemical System	Lithium
Typical Capacity (mAh)	170
Height (mm)	10.80
Diameter (mm)	11.60
Weight (g)	3.00

16.4 COMPUTER INTERFACE

The Printer Module holds a single image: the *current* image of the PenPrint system. Other PenPrint modules read the current image from the Printer Module, while others write a new current image to the Printer Module. Some modules have both read and write ability. The image transfer is accomplished by the PenPrint Serial Bus.

In addition, the USB Module makes all the PenPrint modules visible to an external computer system. This includes the Printer Module with its current image. A computer user can read the current image from the Printer Module or write a new image to the Printer Module for subsequent printing.

16.5 INTERNALS

The Printer Module consists of:

- standard PenPrint male/female bayonet connectors
- a 2-inch Memjet printer (see page 87)
- a replaceable ink cartridge
- an ASIC containing controlling logic and an image store (Flash memory)
- a QA chip [4, 5]
- a 3V power supply

Figure 38 shows an exploded view of the Printer Module, and Figure 39 shows the cross sectional view. A detailed discussion of the Memjet printer can be found on page 87.

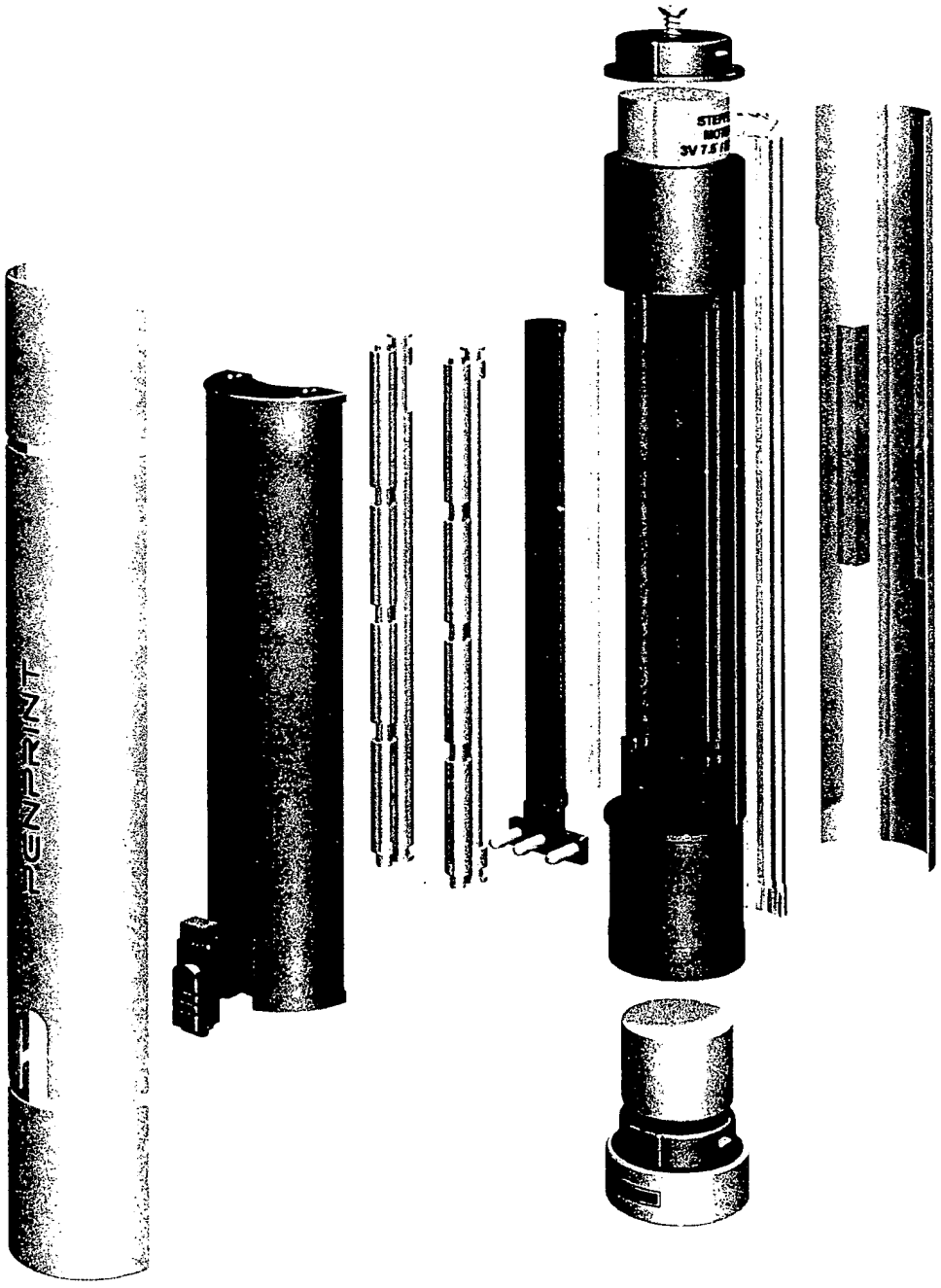
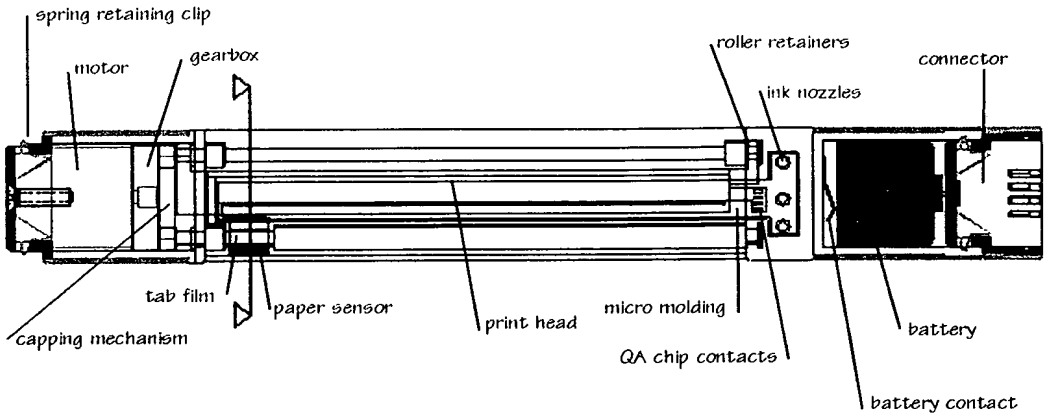


Figure 38. Exploded View of Printer Module



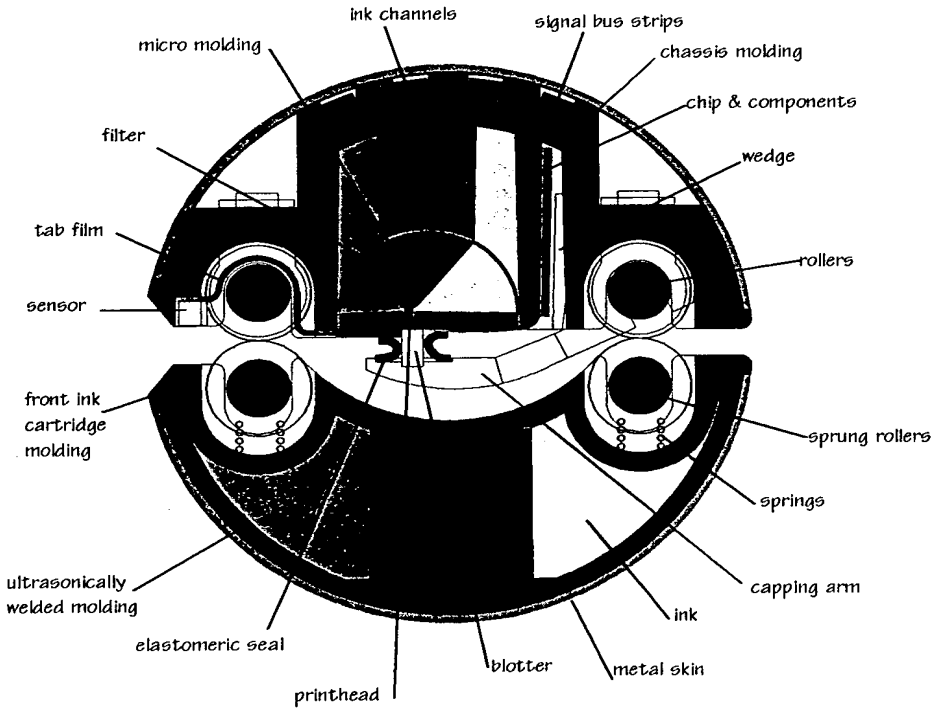


Figure 39. Cross Sectional View of Print Module

17 Image Processing Requirements

There are a number of steps involved in taking an image from ImageRAM and producing a high quality output print. The PenPrint Printer Module is responsible for ensuring that the final output print is as high quality as possible.

From the highest level point of view, there is a single image processing chain. The image is taken from the Image RAM, processed, and sent to the Memjet printer to be printed out.

We describe the high level process as the Image Print Chain, containing a number of steps:

This section describes an *implementation independent* image processing chain that meets the quality requirements of PenPrint. At this stage, we are not considering *exactly how* the processing is performed in terms of hardware, but rather *what* must be done. These functions must be mapped onto various units within the PPCP (see Section 18 on page 62).

17.1 CONSTRAINTS

Regardless of the PPCP implementation, there are a number of constraints:

- The input image is a 267ppi 534 × 850 contone CMY or L*a*b* image.
- The output image is a 1600dpi 3200 × 5100 dithered bi-level CMY image destined for a Memjet printhead.

17.1.1 Timing

PenPrint is primarily bound by power constraints, and consequently uses the low-speed print mode of the Memjet printhead, where an entire line is printed in 200µs. Each PenPrint page is 2 inches × 3.2 inches. Since the Memjet printhead prints at 1600dpi, a page is 5100 lines of 3200 dots per line. Printing 5100 lines at 200µs per line gives a total print time of 1.02 seconds.

In low-speed print mode the Memjet printhead prints an entire line in 200µs. Since all segments fire at the same time 48 nozzles are fired simultaneously with each firing pulse.

Within the printhead interface, a single Print Cycle and a single Load Cycle must both complete within the 200µs line time. In addition, the paper must advance by about 16µm in the same time.

The 800 SRClock pulses to the printhead, each clock pulse transferring 12 bits of valid data, must also take place within the 200µs line time. The length of an SRClock pulse cannot exceed 200µs / 800 = 250ns. The printhead must therefore be clocked at **4MHz**.

The dot calculations for a print line must complete within the 200µs line time. A single line consists of 3200 positions, each with 3 colors. The total number of dots is therefore 9600. Since the dot calculating logic is capable of calculating a single dot per cycle, it implies a clock speed of 9600/200µs = **48MHz** (an integer multiple of the printhead speed).

17.2 IMAGE PRINT CHAIN

The Image Print Chain is concerned with taking an existing image from memory and printing it to a Memjet printer. There are a number of steps required in the image processing chain in order to produce high quality 1600dpi prints from 267ppi images. Figure 76 illustrates the Image Print Chain.

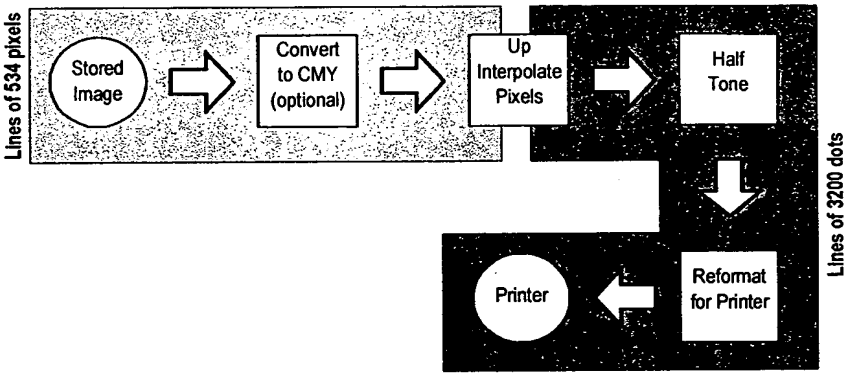


Figure 40. Image Print Chain

17.2.1 Convert from L*a*b* to CMY

The printer's CMY color space does not have a linear response. This is definitely true of pigmented inks, and partially true for dye-based inks. The individual color profile of a particular device (input and output) can vary considerably. Consequently, to allow for accurate conversion, as well as to allow for future sensors, inks, and printers, the L*a*b* color model is used for PenPrint, and the Printer Module has the capability to convert from the well defined and perceptually linear L*a*b* to the particular peculiarities of its CMY color space. Additionally, there is the possibility of a PC performing high quality color space conversion before downloading an image to the PenPrint module.

The stored image is therefore defined in terms of the L*a*b* or CMY color space. If it is in the L*a*b* color space, it must be converted to CMY before being printed out. Rather than convert the L*a*b* to CMY in situ, the conversion is done on the fly during the print. This allows the L*a*b* image to be exported from the Printer Module in a portable color space format. If the image is already in CMY (for example, downloaded from a PC with CMY generated by the PC), the color conversion step is bypassed

The transformations required for color conversion are shown in Figure 41.

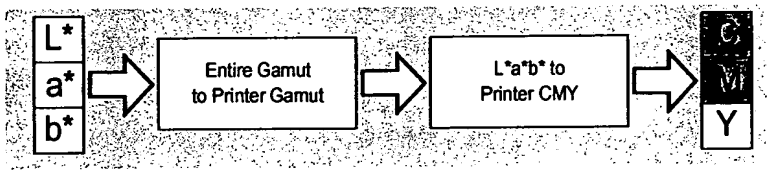


Figure 41. Transformations in Conversion from L*a*b* to CMY

Rather than perform these transformations exhaustively, excellent results can be obtained via a tri-linear conversion based on 3 sets of 3D lookup tables. The lookup tables contain the resultant transformations for the specific entry as indexed by L*a*b*. Three tables are

required: one mapping L*a*b* to C, one mapping L*a*b* to M, and one mapping L*a*b* to Y. Tri-linear interpolation can be used to give the final result for those entries not included in the tables. The process is shown in Figure 42.

Tri-linear interpolation requires reading 8 values from the lookup table, and performing 7 linear interpolations (4 in the first dimension, 2 in the second, and 1 in the third). High precision can be used for the intermediate values, although the output value is only 8 bits.

The size of the lookup table required depends on the linearity of the transformation. The recommended size for each table in this application is 17 × 17 × 17[1], with each entry 8 bits. A 17 × 17 × 17 table is 4913 bytes (less than 5KB).

To index into the 17-per-dimension tables, the 8-bit input color components are treated as fixed-point numbers (4:4). The 4 bits of integer give the index, and the 4 bits of fraction are used for interpolation.

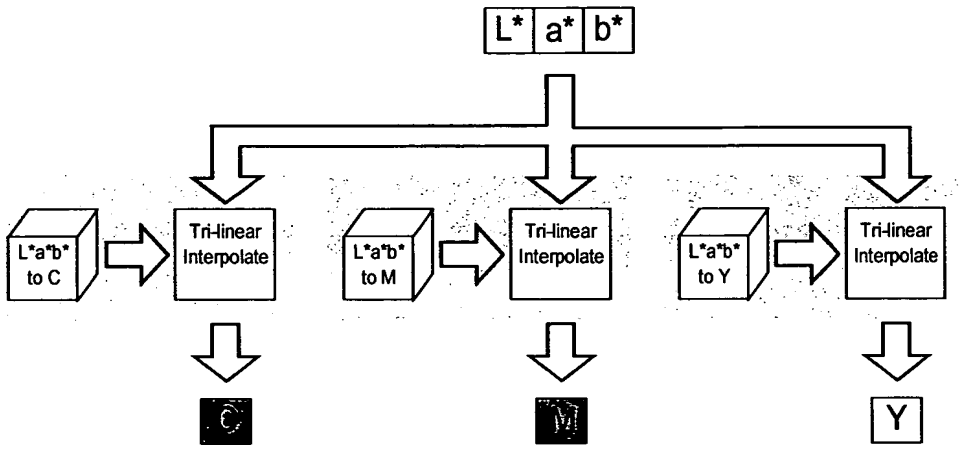


Figure 42. Conversion from RGB to CMY by Trilinear Interpolation

17.2.2 Up Interpolate

The 534 × 850 CMY image must now be up-interpolated to the final print resolution (3200 × 5100). The ratio is 1:6 in both dimensions.

Although it is certainly possible to bi-linearly interpolate the 36 values (1:6 in both X and Y dimensions), the resultant values will not be printed contone. The results will be dithered and printed bi-level. Given that the contone 1600 dpi results will be converted into dithered bi-level dots, the accuracy of bi-linear interpolation from 267 dpi to 1600 dpi will not be visible (the image resolution was chosen for this very reason). Pixel replication will therefore produce good results.

Pixel replication simply involves taking a single pixel, and using it as the value for a larger area. In this case, we replicate a single pixel to 36 pixels (a 6×6 block). If each pixel were contone, the result may appear blocky, but since the pixels are to be dithered, the effect is

1. Although a 17 × 17 × 17 table will give excellent results, it may be possible to get by with only a 9 × 9 × 9 conversion table (729 bytes). The exact size can be determined by simulation. The 5K conservative-but-definite-results approach was chosen for the purposes of this document.

that the 36 resultant bi-level dots take on the contone value. The process is shown in Figure 43.

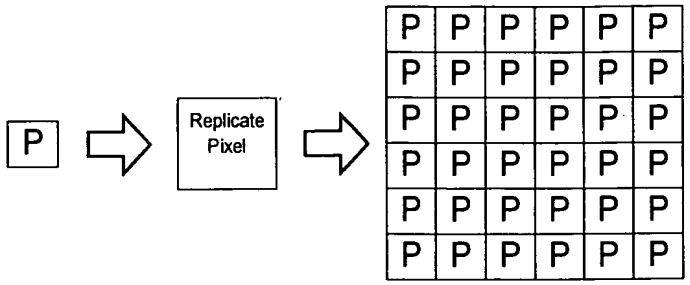


Figure 43. Pixel Replication of a Single Pixel to a 6×6 block

17.2.3 Halftone

The printhead is only capable of printing dots in a bi-level fashion. We must therefore convert from the contone CMY to a dithered CMY image. More specifically, we produce a dispersed dot ordered dither using a stochastic dither cell, converting a contone CMY image into a dithered bi-level CMY image.

The 8-bit 1600 dpi contone value is compared to the current position in the dither cell. If the 8-bit contone value is greater than the dither cell value, an output bit of 1 is generated. Otherwise an output bit of 0 is generated. This output bit will eventually be sent to the printhead and control a single nozzle to produce a single C, M, or Y dot. The bit represents whether or not a particular nozzle will fire for a given color and position.

The same position in the dither cell can be used for C, M, and Y. This is because the actual printhead produces the C, M, and Y dots for different lines in the same print cycle. The staggering of the different colored dots effectively gives us staggering in the dither cell.

The half-toning process can be seen in Figure 44.

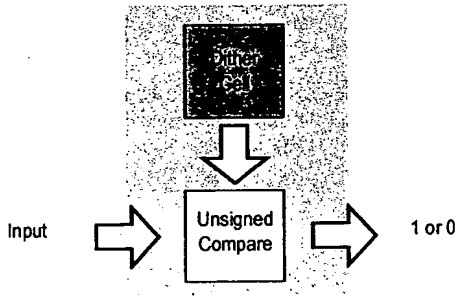


Figure 44. Half-toning using a 50 × 50 Dither Cell

The size of the dither cell depends on the resolution of the output dots. Since we are producing 1600 dpi dots, the cell size should be larger than 32 × 32. In addition, to allow the dot processing order to match the printhead segments, the size of the dither cell should ideally divide evenly into 800 (since there are 800 dots in each segment of the printhead).

A dither cell size of 50 × 50 is large enough to produce high quality results, and divides evenly into 800 (16 times). Each entry of the dither cell is 8 bits, for a total of 2500 bytes (approximately 2.5 KB).

17.2.4 Reformat for Printer

The final process before being sent to the printer is for the dots to be formatted into the correct order for being sent to the printhead. The dots must be sent to the printhead in the correct order - 12 dots at a time for a 2 inch printhead as described in Section 21.1 on page 100.

The dots are produced in the correct order for printing by the up-interpolate and dither functions. Those dot values (each value is 1 bit) can simply be collected, and sent off in groups of 12. The process is shown in Figure 45.

Figure 45. Reformat Dots for Printer

The 12 bit groups can then be sent to the printhead by the Memjet Interface.

18 PenPrint Central Processor (PPCP)

This chapter defines the composition of the PenPrint Central Processor (PPCP) as found in the PenPrint Printer Module.

The PPCP is designed to be fabricated using a 0.25 micron CMOS process, with approximately 9 million transistors, almost half of which are flash memory or static RAM. This leads to an estimated area of 16mm^2. The estimated manufacturing cost is $4 in the year 2001. The PPCP is a relatively straightforward design, and design effort can be reduced by the use of datapath compilation techniques, macrocells, and IP cores. The PPCP contains:

- a low speed CPU/microcontroller core
- 8 KByte flash memory for program storage
- 2 KByte RAM for program variable storage
- a PenPrint Serial Interface
- a parallel interface
- 2 QA Chip interfaces
- 1.3MB multilevel flash memory (2 bits per cell) for image storage
- Image Access Unit
- Printhead Interface

The PPCP is intended to run at a clock speed of approximately 48 MHz on 3V externally and 1.5V internally to minimize power consumption. The actual operating frequency will be an integer multiple of the PenPrint Serial Bus operating frequency. The CPU is intended to be a simple micro-controller style CPU, running at about 1 MHz, and should be a vendor supplied core.

18.1 CPU Core and Memory

18.1.1 CPU Core

The PPCP incorporates a simple micro-controller CPU core to synchronize the image capture and printing image processing chains and to perform Printcam's general operating system duties including the user-interface. A wide variety of CPU cores are suitable: it can be any processor core with sufficient processing power to perform the required calculations and control functions fast enough to met consumer expectations.

Since all of the image processing is performed by dedicated hardware, the CPU does not have to process pixels. As a result, the CPU can be extremely simple. However it must be fast enough to run the stepper motor during a print (the stepper motor requires a 5KHz process). An example of a suitable core is a Philips 8051 micro-controller running at about 1 MHz.

There is no need to maintain instruction set continuity between different PenPrint models. Different PPCP chip designs may be fabricated by different manufacturers, without requiring to license or port the CPU core. This device independence avoids the chip vendor lock-in such as has occurred in the PC market with Intel.

Associated with the CPU Core is a Program ROM and a small Program Scratch RAM.

The CPU communicates with the other units within the PPCP via memory-mapped I/O. Particular address ranges map to particular units, and within each range, to particular registers within that particular unit. This includes the serial and parallel interfaces.

18.1.2 Program ROM

A small Program Flash ROM is incorporated into the PPCP. The ROM size depends on the CPU chosen, but should not be more than 8KB.

18.1.3 Program RAM

Likewise, a small scratch RAM area is incorporated into the PPCP. Since the program code does not have to manipulate images, there is no need for a large scratch area. The RAM size depends on the CPU chosen (e.g. stack mechanisms, subroutine calling conventions, register sizes etc.), but should not be more than about 2KB.

18.1.4 CPU Memory Decoder

The CPU Memory Decoder is a simple decoder for satisfying CPU data accesses. The Decoder translates data addresses into internal PPCP register accesses over the internal low speed bus, and therefore allows for memory mapped I/O of PPCP registers.

18.2 COMMUNICATION INTERFACES

18.2.1 PenPrint Serial Port Interface

This is a serial port, connected to the internal chip low-speed bus. The serial port is controlled by the CPU and follows the USB protocol. The serial port allows the transfer of images to and from the PenPrint Printer Module, by external control.

18.2.2 QA Chip Serial Interfaces

These are two standard low-speed serial ports, connected to the internal chip low-speed bus. The CPU-mediated protocol between the two is used to authenticate the ink cartridge [4, 5]. The processor can then retrieve ink characteristics from the QA chip, as well as the remaining volume of each ink. The processor uses the ink characteristics to properly configure the Memjet printhead. It uses the remaining ink volumes, updated on a page-by-page basis with ink consumption information accumulated by the Printhead Interface, to ensure that it never allows the printhead to be damaged by running dry.

The reason for having two ports is to connect to both the on-PenPrint QA Chip and to the ink cartridge's QA Chip using separate lines. The two QA chips are implemented as Authentication Chips [5]. If only a single line is used, a clone ink cartridge manufacturer could usurp the authentication mechanism [4].

18.2.2.1 Ink Cartridge's QA Chip

Each ink cartridge consumable contains its own QA chip. The QA chip contains information required for maintaining the best possible print quality, and is implemented using an Authentication Chip[5]. The 256 bits of data are allocated as follows:

Table 8. Ink Cartridge's 256 bits (16

M[n]	Access	Description
0	RO[a]	Basic Header, Flags etc. (16 bits)
1	RO	Serial number (16 bits)
2	RO	Batch number (16 bits)
3	RO	Reserved for future expansion (must be 0)
4-5	RO	Cyan ink properties (32 bits)
6-7	RO	Magenta ink properties (32 bits)
8-9	RO	Yellow ink properties (32 bits)
10-11	DO[b]	Cyan ink remaining in nanolitres (32 bits)
12-13	DO	Magenta ink remaining in nanolitres (32 bits)
14-15	DO	Yellow ink remaining in nanolitres (32 bits)

a. Read Only

b. Decrement Only

Before each print, the amount of ink remaining is checked by the CPU to ensure that there is enough for a worst-case page (a full coverage print). Once the image has been printed, the processor multiplies the total number of drops of each color (obtained from the Printhead Interface) by the drop volume. The amount of printed ink is subtracted from the amount of ink remaining. The unit of measurement for ink remaining is nanolitres, so 32 bits can represent over 4 liters of ink (even though a single ink cartridge only holds about 2.7mls of ink of each color). The amount of ink used for a page must be rounded up to the nearest nanolitre (i.e. approximately 1000 printed dots)

18.2.3 Parallel Interface

The parallel interface connects the PPCP to individual static electrical signals. The CPU is able to control each of these connections as memory-mapped I/O via the low-speed bus. (See Section 18.1.4 on page 63 for more details on memory-mapped I/O).

Table 9 shows the connections to the parallel interface.

Table 9. Connections to Parallel Interface

Connection	Direction	Pins
Paper transport stepper motor	Out	4
Capping solenoid	Out	1
Buttons	In	2
	TOTAL	7

18.2.4 JTAG Interface

A standard JTAG (Joint Test Action Group) Interface is included in the PPCP for testing purposes. Due to the complexity of the chip, a variety of testing techniques are required, including BIST (Built In Self Test) and functional block isolation. An overhead of 10% in chip area is assumed for overall chip testing circuitry.

18.3 IMAGE RAM

The Image RAM is used to store the current print image. The Image RAM is multi-level Flash (2-bits per cell) so that the image is retained after the power has been shut off.

The image held in Image RAM is kept in a interleaved format (the color components for a given pixel are stored together). Images are stored in one of two formats: either as CMY or L*a*b*, with each image represented by 850 lines containing 534 sets of 3 8-bit color samples each.

The total amount of memory required for the interleaved linear CMY/L*a*b* image is 1,361,700 bytes (approximately 1.3 MB).

The image is written to Image RAM by the Image Access Unit, and read by both the Image Access Unit and the Print Generator Unit. The CPU does not have direct random access to this image memory. It must access the image pixels via the Image Access Unit.

18.4 IMAGE ACCESS UNIT

The Image Access Unit (IAU) produces the means for the CPU to access the image in ImageRAM. The CPU can read pixels from the image in ImageRAM and write pixels back. The IAU allows planar and interleaved access. It also allows for 90 degrees rotation. The conceptual difference between rotated and non-rotated images can be seen in Figure 46. Note that the non-rotated is with reference to the final print image.

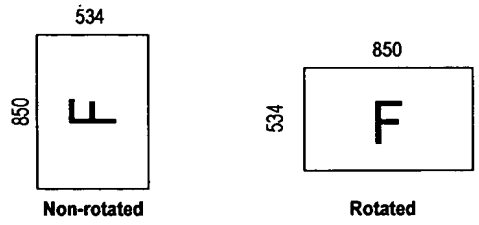


Figure 46. Conceptual Rotated and Non-rotated Image Access

Pixels are read by the CPU when the image is to be transferred to another PenPrint Module. Pixels are written by the CPU when the image is being loaded from another PenPrint Module. The registers of the IAU allow pixel transfers to be planar/interleaved or rotated by 90 degrees as desired.

The Image Access Unit is a straightforward access mechanism to ImageRAM, and operates via the register set as shown in Table 10.

Table 10. IAU Registers

Name	Bits	Description
ImageAddress	21	Address to read or write in ImageRAM
Delta12	12	Amount to add to ImageAddress when stepping from one pixel to the next in reads/writes for the first 2 of each set of 3.
Delta2	12	Amount to add to ImageAddress when stepping from one pixel to the next in reads/writes during the 3rd access of each set of 3.
Mode	3	0 = Read from ImageAddress into Value. 1 = Write Value to ImageAddress.
Value	8	Value stored at ImageAddress (if Mode = Read) Value to store at ImageAddress (if Mode = Write)

The data is read from or written to the appropriate address in Image RAM whenever the Value register is read from or written to, according to the sense of Mode. Interleaved/planar and rotated access is accomplished via the two Delta registers. The values should be set as shown in Table 11:

Table 11. Register Settings for Different Image Access Modes

Access Type	Image Read or Written as	Image Address	Delta12	Delta3
Interleaved (each pixel 3 colors)	850 rows × 534 pixels	0	1	1
	534 rows × 850 pixels	0	1	1600[a]
Planar - plane N (each pixel 1 color)	850 rows × 534 pixels	N	3	3
	534 rows × 850 pixels	N	1602[b]	1602

a. 534×3-2

b. 534×2

The structure of the Image Access Unit is very simple, as shown in Figure 47.

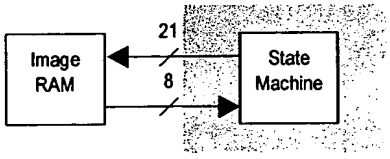


Figure 47. Image Access Unit

18.5 PRINTHEAD INTERFACE

The Printhead Interface (PHI) is the means by which the PPCP loads the Memjet printhead with the dots to be printed, and controls the actual dot printing process. The PHI is a wrapper for a number of units, namely:

- a Memjet Interface (MJI), which transfers data to the Memjet printhead, and controls the nozzle firing sequences during a print.

- a LineSyncGen unit (LSGU), which provides synchronization signals for the MJI and the stepper motors.
- a Print Generator Unit (PGU) is an implementation of most of the Print Chain described in Section 17.2 on page 58, as well as providing a means of producing test patterns. The PGU takes a image from the ImageRAM (in L*a*b* or CMY) and produces a 1600 dpi dithered CMY image in real time as required by the Memjet Interface. In addition, the PGU has a Test Pattern mode, which enables the CPU to specify precisely which nozzles are fired during a print.

The units within the PHI are controlled by a number of registers that are programmed by the CPU.

The internal structure of the Printhead Interface is shown in Figure 48.

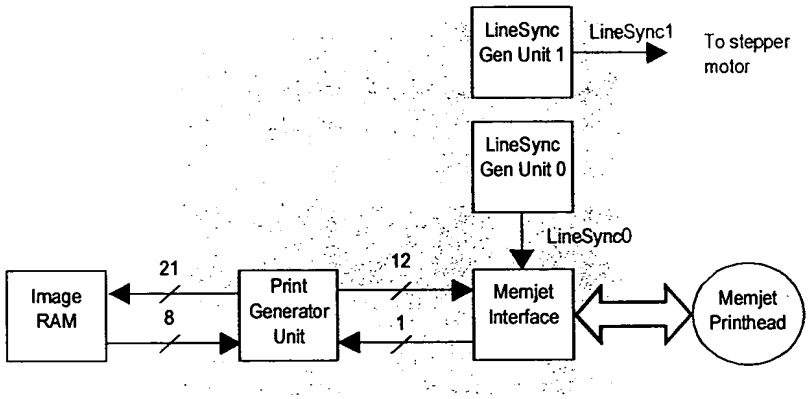


Figure 48. Internal Structure Printhead Interface

In the PHI there are two LSGUs. The first LSGU produces LineSync0, which is used to control the Memjet Interface. The second LSGU produces LineSync1 which is used to pulse the paper drive stepper motor.

The following sections detail the LineSyncGen Unit, the Memjet Interface and the Print Generator Unit respectively.

18.5.1 LineSyncGen Unit

The LineSyncGen units (LSGU) are responsible for generating the synchronization pulses required for printing a page. Each LSGU produces an external LineSync signal to enable line synchronization. The generator inside the LGSU generates a LineSync pulse when told to 'go', and then every so many cycles until told to stop. The LineSync pulse defines the start of the next line.

The exact number of cycles between LineSync pulses is determined by the CyclesBetweenPulses register, one per generator. It must be at least long enough to allow one line to print (200 µs for the low speed printing) and another line to load, but can be longer as desired (for example, to accommodate special requirements of paper transport circuitry). If the CyclesBetweenPulses register is set to a number less than a line print time, the page will not print properly since each LineSync pulse will arrive before the particular line has finished printing.

The following interface registers are contained in the LSGU:

Table 12. LineSyncGen Unit Registers

Register Name	Description
CyclesBetweenPulses	The number of cycles to wait between generating one LineSync pulse and the next.
Go	Controls whether the LSGU is currently generating LineSync pulses or not. A write of 1 to this register generates a LineSync pulse, transfers CyclesBetweenPulses to CyclesRemaining, and starts the countdown. When CyclesRemaining hits 0, another LineSync pulse is generated, CyclesBetweenPulses is transferred to CyclesRemaining and the countdown is started again. A write of 0 to this register stops the countdown and no more LineSync pulses are generated.
CyclesRemaining	A status register containing the number of cycles remaining until the next LineSync pulse is generated.

18.5.2 Memjet Interface

The Memjet Interface (MJI) connects the PPCP to the external Memjet printhead (see "Memjet Printhead" on page 87.), providing both data and appropriate signals to control the nozzle loading and firing sequences during a print.

The Memjet Interface is simply a State Machine (see Figure 49) which follows the printhead loading and firing order described in Section 20.2 on page 93, and includes the functionality of the Preheat cycle and Cleaning cycle as described in Section 20.4 on page 97 and Section 20.5 on page 97.

The MJI loads data into the printhead from a choice of 2 data sources:

- All 1s. This means that all nozzles will fire during a subsequent Print cycle, and is the standard mechanism for loading the printhead for a Preheat or Cleaning cycle.
- From the 12-bit input held in the Transfer register of the PGU. This is the standard means of printing an image, whether it be a photo or test pattern. The 12-bit value from the PGU is directly sent to the printhead and a 1-bit 'Advance' control pulse is sent to the PGU.

The MJI knows how many lines it has to print for the page. When the MJI is told to 'go', it waits for a LineSync pulse before it starts the first line. Once it has finished loading/printing a line, it waits until the next LineSync pulse before starting the next line. The MJI stops once the specified number of lines has been loaded/printed, and ignores any further LineSync pulses.

The MJI must be started after the PGU has already prepared the first 12-bit transfer value. This is so the 12-bit data input will be valid for the first transfer to the printhead.

The MJI is therefore directly connected to the PGU, LineSync0, and the external Memjet printhead. The basic structure is shown in Figure 49.

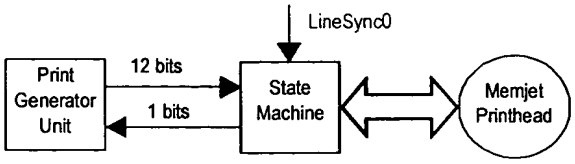


Figure 49. Memjet Interface

18.5.2.1 Connections to Printhead

The MJI has the following connections to the printhead, with the sense of input and output with respect to the MJI. The names match the pin connections on the printhead (see Section 21.4 on page 102 for an explanation of the pins).

Table 13. Printhead Connections

Name	#Pins	I/O	Description
ChromapodSelect	4	O	Select which chromapod will fire (0-9)
NozzleSelect	4	O	Select which nozzle from the pod will fire (0-9)
AEnable	1	O	Firing pulse for phasegroup A
BEnable	1	O	Firing pulse for phasegroup B
CDataIn[0-3]	4	O	Cyan output to cyan shift register of segments 0-3
MDataIn[0-3]	4	O	Magenta input to magenta shift register of segments 0-3
YDataIn[0-3]	4	O	Yellow input to yellow shift register of segments 0-3
SRClock	1	O	A pulse on SRClock (ShiftRegisterClock) loads the current values from CDataIn[0-3], MDataIn[0-3] and YDataIn[0-3] into the 12 shift registers of the printhead
PTransfer	1	O	Parallel transfer of data from the shift registers to the printhead's internal NozzleEnable bits (one per nozzle).
SenseSegSelect	1	O	A pulse on SenseSegSelect ANDed with data on CDataIn[n] selects the sense lines for segment n.
Tsense	1	I	Temperature sense
Vsense	1	I	Voltage sense
Rsense	1	I	Resistivity sense
Wsense	1	I	Width sense
TOTAL	29		

18.5.2.2 Firing Pulse Duration

The duration of firing pulses on the AEnable and BEnable lines depend on the viscosity of the ink (which is dependent on temperature and ink characteristics) and the amount of power available to the printhead. The typical pulse duration range is 1.3 to 1.8 µs. The MJI therefore contains a programmable pulse duration table, indexed by feedback from the printhead. The table of pulse durations allows the use of a lower cost power supply, and aids in maintaining more accurate drop ejection.

The Pulse Duration table has 256 entries, and is indexed by the current Vsense and Tsense settings. The upper 4-bits of address come from Vsense, and the lower 4-bits of address come from Tsense. Each entry is 8 bits, and represents a fixed point value in the range of 0-4µs. The process of generating the AEnable and BEnable lines is shown in Figure 50.

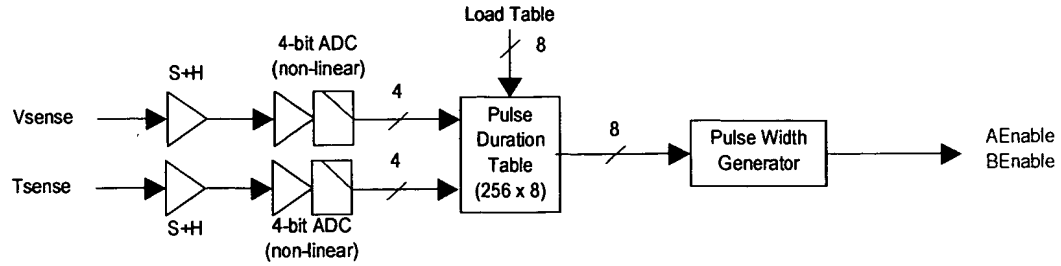


Figure 50. Generation of AEnable and BEnable Pulse Widths

The 256-byte table is written by the CPU before printing the image. Each 8-bit pulse duration entry in the table combines:

- Brightness settings
- Viscosity curve of ink (from the QA Chip)
- Rsense
- Wsense
- Tsense
- Vsense

18.5.2.3 Dot Counts

The MJI maintains a count of the number of dots of each color fired from the printhead. The dot count for each color is a 24-bit value, individually cleared under processor control. Each dot count can hold a maximum coverage dot count of a single 3-inch print, so in typical usage, the dot count will be read and cleared after each print.

The dot counts are used by the CPU to update the QA chip (see Section 18.2.2.1 on page 64) in order to predict when the ink cartridge runs out of ink. The processor knows the volume of ink in the cartridge for each of C, M, and Y from the QA chip. Counting the number of drops eliminates the need for ink sensors, and prevents the ink channels from running dry. An updated drop count is written to the QA chip after each print. A new image will not be printed unless there is enough ink left, and allows the user to change the ink without getting a dud photo which must be reprinted.

The layout of the dot counter for cyan is shown in Figure 51. The remaining 2 dot counters (MDotCount and YDotCount, for magenta and yellow respectively) are identical in structure.

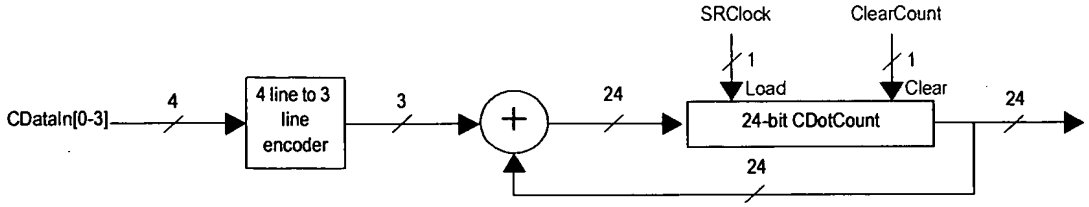


Figure 51. Dot Count Logic

18.5.2.4 Registers

The CPU communicates with the MJI via a register set. The registers allow the CPU to parameterize a print as well as receive feedback about print progress.

The following registers are contained in the MJI:

Table 14. Memjet Interface Registers

Register Name	Description
Print Parameters	
NumTransfers	The number of transfers required to load the printhead (usually 800). This is the number of pulses on the SRClock and the number of 12-bit data values to transfer for a given line.
NumLines	The number of Load/Print cycles to perform.
Monitoring the Print	
Status	The Memjet Interface's Status Register
LinesRemaining	The number of lines remaining to be printed. Only valid while Go=1. Starting value is NumLines and counts down to 0.
TransfersRemaining	The number of transfers remaining before the Printhead is considered loaded for the current line. Only valid while Go=1. Starting value is NumTransfers and counts down to 0.
SenseSegment	The 4-bit value to place on the Cyan data lines during a subsequent feedback SenseSegSelect pulse. Only 1 of the 4 bits should be set, corresponding to one of the 4 segments.
SetAllNozzles	If non-zero, the 12-bit value written to the printhead during the LoadDots process is all 1s, so that all nozzles will be fired during the subsequent PrintDots process. This is used during the preheat and cleaning cycles. If 0, the 12-bit value written to the printhead comes from the Print Generator Unit. This is the case during the actual printing of a photo or test images.
Actions	
Reset	A write to this register resets the MJI, stops any loading or printing processes, and loads all registers with 0.

Table 14. Memjet Interface Registers

Register Name	Description
SenseSegSelect	A write to this register with any value clears the FeedbackValid bit of the Status register, and the remaining action depends on the values in the LoadingDots and PrintingDots status bits. If either of the status bits are set, the Feedback bit is cleared and nothing more is done. If both status bits are clear, a pulse is given simultaneously on the SenseSegSelect line with all Cyan data bits 0. This stops any existing feedback. A pulse is then given on SenseSegSelect with the Cyan data bits set according to the SenseSegment register. Once the various sense lines have been tested, the values are placed in the Tsense, Vsense, Rsense, and Wsense registers, and the Feedback bit of the Status register is set.
Go	A write of 1 to this bit starts the LoadDots / PrintDots cycles, which commences with a wait for the first LineSync pulse. A total of NumLines lines are printed, each line being loaded/printed after the receipt of a LineSync pulse. The loading of each line consists of NumTransfers 12-bit transfers. As each line is printed, LinesRemaining decrements, and TransfersRemaining is reloaded with NumTransfers again. The status register contains print status information. Upon completion of NumLines, the loading/printing process stops, the Go bit is cleared, and any further LineSync pulses are ignored. During the final print cycle, nothing is loaded into the printhead. A write of 0 to this bit stops the print process, but does not clear any other registers.
ClearCounts	A write to this register clears the CDotCount, MDotCount, and YDotCount, registers if bits 0, 1, or 2 respectively are set. Consequently a write of 0 has no effect.
Feedback	
Tsense	Read only feedback of Tsense from the last SenseSegSelect pulse sent to segment SenseSegment. Is only valid if the FeedbackValid bit of the Status register is set.
Vsense	Read only feedback of Vsense from the last SenseSegSelect pulse sent to segment SenseSegment. Is only valid if the FeedbackValid bit of the Status register is set.
Rsense	Read only feedback of Rsense from the last SenseSegSelect pulse sent to segment SenseSegment. Is only valid if the FeedbackValid bit of the Status register is set.
Wsense	Read only feedback of Wsense from the last SenseSegSelect pulse sent to segment SenseSegment. Is only valid if the FeedbackValid bit of the Status register is set.
CDotCount	Read only 24-bit count of cyan dots sent to the printhead.
MDotCount	Read only 24-bit count of magenta dots sent to the printhead.
YDotCount	Read only 24-bit count of yellow dots sent to the printhead.

The MJI's Status Register is a 16-bit register with bit interpretations as follows:

Table 15. MJI Status Register

Name	Bits	Description
LoadingDots	1	If set, the MJI is currently loading dots, with the number of dots remaining to be transferred in TransfersRemaining. If clear, the MJI is not currently loading dots
PrintingDots	1	If set, the MJI is currently printing dots. If clear, the MJI is not currently printing dots.
PrintingA	1	This bit is set while there is a pulse on the AEnable line
PrintingB	1	This bit is set while there is a pulse on the BEnable line
FeedbackValid	1	This bit is set while the feedback values Tsense, Vsense, Rsense, and Wsense are valid.
Reserved	3	-
PrintingChromapod	4	This holds the current chromapod being fired while the PrintingDots status bit is set.
PrintingNozzles	4	This holds the current nozzle being fired while the Printing-Dots status bit is set.

The following pseudocode illustrates the logic required to load a printhead for a single line. Note that loading commences only after the LineSync pulse arrives. This is to ensure the data for the line has been prepared by the PGU and is valid for the first transfer to the printhead.

```
Wait for LineSync
For TransfersRemaining = NumTransfers to 0
   If (SetAllNozzles)
      Set all ColorData lines to be 1
   Else
      Place 12 bit input on 12 ColorData lines
   EndIf
   Pulse SRClock
   Wait 12 cycles
   Send ADVANCE signal
EndFor
```

18.5.2.5 Preheat and Cleaning Cycles

The Cleaning and Preheat cycles are simply accomplished by setting appropriate registers in the MJI:

- SetAllNozzles = 1
- Set the PulseDuration register to either a low duration (in the case of the preheat mode) or to an appropriate drop ejection duration for cleaning mode.
- Set NumLines to be the number of times the nozzles should be fired
- Set the Go bit and then wait for the Go bit to be cleared when the print cycles have completed.

The LSGU must also be programmed to send LineSync pulses at the correct frequency.

18.5.3 Print Generator Unit

The Print Generator Unit (PGU) is an implementation of the Print Chain described in Section 17.2 on page 58, as well as providing a means of producing test patterns.

From the simplest point of view, the PGU provides the interface between the Image RAM and the Memjet Interface, as shown in Figure 52. The PGU takes a planarized linear RGB obtained from a CFA format captured image from the ImageRAM, and produces a 1600 dpi dithered CMY image in real time as required by the Memjet Interface. In addition, the PGU has a Test Pattern mode, which enables the CPU to specify precisely which nozzles are fired during a print. The MJI provides the PGU with an Advance pulse once the 12-bits have been used.

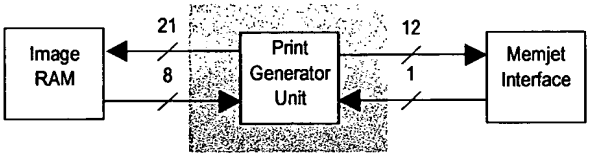


Figure 52. Interface of Print Generator Unit

The PGU has 2 image processing chains. The first, the Test Pattern mode, simply reads data directly from Image RAM, and formats it in a buffer ready for output to the MJI. The second contains the majority of Print Chain functions (see Section 17.2 on page 58):

- Convert L*a*b* to CMY
- Up-Interpolate
- Halftone
- Reformat for Printer

The PGU takes as input a variety of parameters, including L*a*b* to CMY conversion tables and printing timing parameters.

The two process chains can be seen in Figure 53, where data flow is shown (address and register enable flags are not shown). The most direct chain goes from the Image RAM to Buffer 1 via the Test Pattern Access process. The other chain consists of 2 processes, all running in parallel. The first process performs color conversion while the second performs the up-interpolation, halftoning, and reformatting for the printer. The processes are connected via buffers.

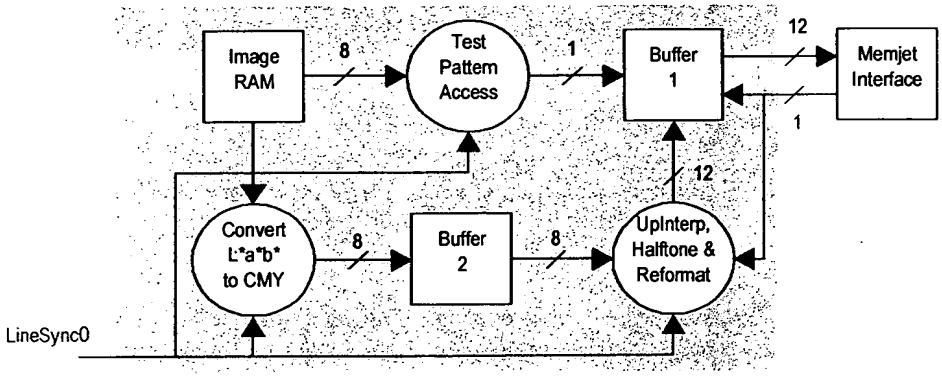


Figure 53. Print Generator Unit (shaded area)

The buffer sizes are shown in Table 16.

Table 16. Buffer sizes for Print Generator Unit

Buffer	Size (bytes)	Composition of Buffer
Buffer 1	5	3 × 12 bits
Buffer 2	9,612	3 colors(CMY) × 6 lines × 534 contone pixels @ 8-bits each
TOTAL	9,617	

Apart from a number of registers, some of the processes have significant lookup tables or memory components. These are summarized in Table 17.

Table 17. Memory requirements within PGU Processes

Unit	Size (bytes)	Composition of Requirements
Test Pattern Access	0	
Convert L*a*b* to CMY	14,739	3 conversion tables, each 17×17×17×8-bits
UpInterpolate / Halftone / Reformat	2,500	Dither Cell, 50×50×8-bits
TOTAL	17,239	

18.5.3.1 Buffer 1

Buffer 1 holds the generated dots from the entire Print Generation process. Buffer 1 consists of a 12-bit shift register to hold dots generated one at a time from the UHRU (UpInterpolate-Halftone and Reformat Unit), 3 4-bit registers to hold the data generated from the TPAU (Test Pattern AccessUnit), and a 12-bit register used as the buffer for data transfer to the MJI (Memjet Interface). A pulse on either the Advance line from the MJI, or the TransferWriteEnable from both the TPAU and the UHRU, loads the 12-bit Transfer register with all 12-bits, either from the 3 4-bit registers or the single 12-bit shift register.

Buffer 1 therefore acts as a double buffering mechanism for the generated dots, and has a structure as shown in Figure 54.

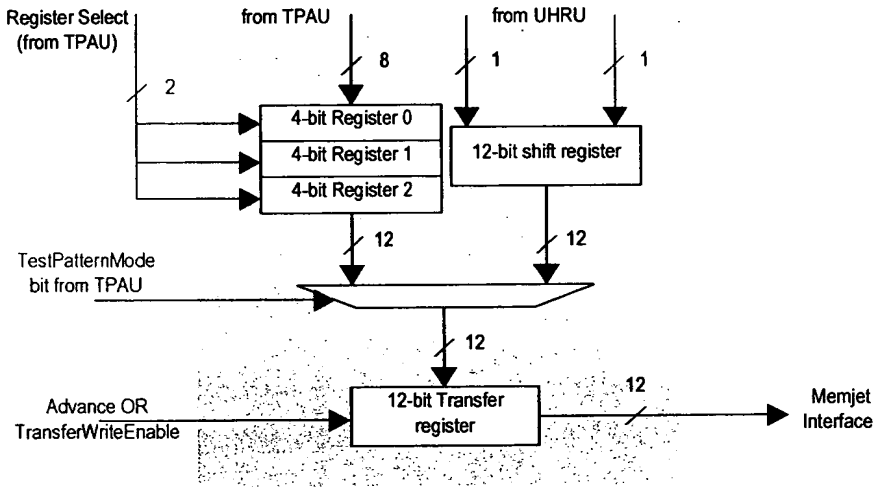


Figure 54. Structure of Buffer 1

A multiplexor chooses between the two 12-bit outputs and sends the result to the 12-bit Transfer register.

The 8-bit input from the TPAU is directed to the 3 4-bit registers by 2 RegisterSelect bits as follows:

- 00 = Write lower 4 bits to register 0, upper 4 bits to register 1
- 01 = Write lower 4 bits to register 2
- 10 = Write upper 4 bits to register 0
- 11 = Write lower 4 bits to register 1, upper 4 bits to register 2

18.5.3.2 Buffer 2

Buffer 2 holds 6 lines of the calculated CMY contone image. Buffer 2 is generated by the Color Conversion process, and accessed by the Up-Interpolate, Halftone and Reformat process in order to generate output dots for the printer.

The size of the Contone Buffer is dependent on the physical distance between the nozzles on the printhead. As dots for one color are being generated for one physical line, dots for a different color on a different line are being generated. The net effect is that 6 different physical lines are printed at the one time from the printer - odd and even dots from different output lines, and different lines per color. This concept is explained and the distances are defined in Section 20.1 on page 89.

The practical upshot is that there is a given distance in Memjet printer space from the even cyan dots through the magenta dots to the odd yellow dots. In order to minimize the conversion from L*a*b* to CMY (since this process takes some time), the CMY pixels that generate those high-res dots are buffered in Buffer 2.

Since the ratio of 534-res lines to 1600dpi lines is 1:6, each contone pixel is sampled 6 times in each dimension. For the purposes of buffer lines, we are only concerned with 1 dimension, so only consider 6 dot lines coming from a single pixel line. The distance

between nozzles of different colors is 4-8 dots (depending on Memjet parameters). We therefore assume 8, which gives a separation distance of 16 dots, or 17 dots in inclusive distance. The worst case scenario is that the 17 dot lines includes the last dot line from a given pixel line. This implies 5 pixel lines, with dot lines generated as 1, 5, 5, 5, 1, and allows an increase of nozzle separation to 10.

To ensure that the contone generation process writing to the buffer does not interfere with the dot generation process reading from the buffer, we add an extra line per color, for a total of 6 lines per color.

The contone buffer is therefore 3 colors of 6 lines, each line containing 534 8-bit contone values. The total memory required is 3 × 6 × 534 = 9,612 bytes (9.5 KBytes). The memory only requires a single 8-bit read per cycle, and a single 8-bit write every 36 cycles (each contone pixel is read 36 times). The structure of Buffer 2 is shown in Figure 55.

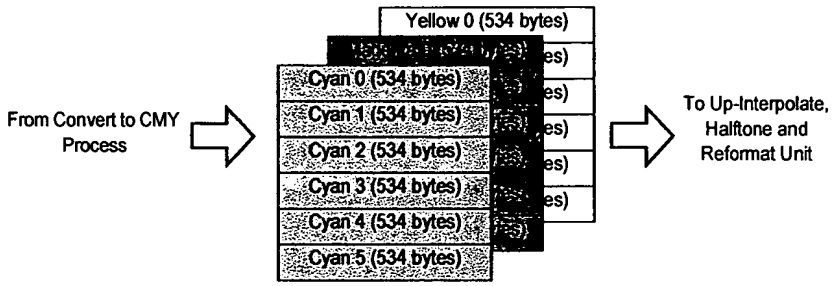


Figure 55. Structure of Buffer 2

Buffer 2 can be implemented as single cycle double access (read and write) RAM running at the nominal speed of the printhead dot generation process.

Buffer 2 is set to white (all 0) before the start of the print process.

18.5.3.3 Test Pattern Access

The Test Pattern Access process is the means by which test patterns are produced. Under normal user circumstances, this process will not be used. It is primarily for diagnostic purposes.

The Test Pattern Access reads the Image RAM and passes the 8-bit values directly to Buffer 1 for output to the Memjet Interface. It does not modify the 8-bit values in any way. The data in the Image RAM would be produced by the CPU using the Image Access Unit (for example, downloaded from a computer via the PenPrint USB Module).

The data read from Image RAM is read in a very simple wraparound fashion. Two registers are used to describe the test data: the start address of the first byte, and the number of bytes. When the end of the data is reached, the data is read again from the beginning.

The structure of the Test Pattern Access Unit is shown in Figure 56.

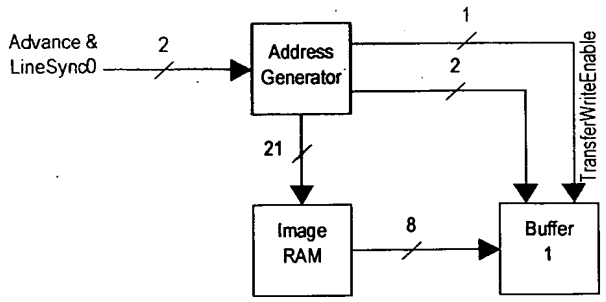


Figure 56. Test Pattern Access Unit

As can be seen in Figure 56, the Test Pattern Access Unit is little more than an Address Generator. When started, the generator reads 2 bytes, waits for an Advance pulse, then reads the 2nd byte again followed by the 3rd byte., and then waits for an Advance pulse. This continues until the LineSync0 pulse, whereupon the process begins again from the current address.

The reason for having a process that reads sets of [byte 1, byte 2, byte 2, byte 3] is that the data from Image RAM is read in 8-bit chunks, while the data passed to the Memjet interface is in 12-bit chunks. Two Advance pulses signify 24 bits, or 3 bytes.

The addresses generated for the Image RAM are based on a start address and a byte count as shown in Table 18.

Table 18. Test Pattern Access Registers

Register Name	Description
TestModeEnabled	If 1, TestMode is enabled. If 0, TestMode is not enabled.
DataStart	Start Address of test data in Image RAM
DataLength	Number of 3 bytes in test data

The following pseudocode illustrates the address generation.

```
Do Forever
   Adr = DataStart
   Remaining = DataLength
   Read Adr into Buffer 1, Adr=Adr+1 (RegisterSelect 00)
   Read Adr into Buffer 1 (RegisterSelect 01)
   Wait for LineSync0 or Advance
   Read Adr into Buffer 1, Adr=Adr+1 (RegisterSelect 10)
   Read Adr into Buffer 1, Adr=Adr+1 (RegisterSelect 11)
   Remaining = Remaining-1
   if (Remaining = 0)
      Remaining = DataLength
   Wait for LineSync0 or Advance
EndDo
```

The RegisterSelect values are derived from Buffer 1 (see Section 18.5.3.1 on page 75).

It is the responsibility of the CPU to ensure that the data is meaningful for the printhead. Byte 0 and the low nybble of byte 1 is the 12-bit nozzle-fire data for the 4 segments cyan, magenta and yellow even nozzles (0, 2, 4 etc.). The upper nybble of byte 1 and byte 2 contains the 12-bit nozzle-fire data for the 4 segments odd nozzles (1, 3, 5 etc.). Note that odd and even nozzles are separated physically by 1 dot line.

18.5.3.4 Convert L*a*b* to CMY

The conversion from L*a*b* to CMY is performed as described in Section 17.2.1 on page 58. The conversion is optional since the input data may already be in CMY format. In the latter case the data is simply passed through with no change.

The conversion process must produce the contone buffer pixels (Buffer 2) at a rate fast enough to keep up with the UpInterpolate-Halftone-Reformat process. Since each contone value is used for 36 cycles (6 times in each of the X and Y dimensions), the conversion process can take up to 36 cycles. This totals 108 cycles for all 3 color components.

The process as described here only requires 14 cycles per color component. The conversion is performed as tri-linear interpolation. Three 17 × 17 × 17 lookup tables are used for the conversion process: L*a*b* to Cyan, L*a*b* to Magenta, and L*a*b* to Yellow. However, since we have 36 cycles to perform each tri-linear interpolation, there is no need for a fast tri-linear interpolation unit. Instead, 8 calls to a linear interpolation process is more than adequate.

Address generation for indexing into the lookup tables is straightforward. We use the 4 most significant bits of each 8-bit color component for address generation, and the 4 least significant bits of each 8-bit color component for interpolating between values retrieved from the conversion tables. The addressing into the lookup table requires an adder due to the fact that the lookup table has dimensions of 17 rather than 16. Fortunately, multiplying a 4-bit number X by 17 is an 8-bit number XX, and therefore does not require an adder or multiplier, and multiplying a 4 bit number by 17^2 (289) is only slightly more complicated, requiring a single add.

Although the interpolation could be performed faster, we use a single adder to generate addresses and have a single cycle interpolation unit. Consequently we are able to calculate the interpolation for generating a single color component from L*a*b* in 14 cycles, as shown in Table 43. The process must be repeated 3 times in order to generate cyan, magenta, and yellow. Faster methods are possible, but not necessary.

Table 19. Trilinear interpolation for color conversion

Cycle	Load	Effective Fetch	Adjust ADR register	Interpolate
1			ADR = 289L*	
2			ADR = ADR +17a*	
3			ADR = ADR + b*	
4	P1	L*a*b*	ADR = ADR + 1	
5	P2	L*a*b*+1	ADR = ADR + 16	
6	P1	L*a*+b*	ADR = ADR + 1	P3 = P1 to P2 by b*
7	P2	L*a*+b*+1	ADR = ADR + 271	
8	P1	L*+a*b*	ADR = ADR + 1	P4 = P1 to P2 by b*
9	P2	L*+a*b*+1	ADR = ADR + 16	P5 = P3 to P4 by a*

Table 19. Trilinear interpolation for color conversion

Cycle	Load	Effective Fetch	Adjust ADR register	Interpolate
10	P1	L*+a*+b*	ADR = ADR + 1	P3 = P1 to P2 by b*
11	P2	L*+a*+b*+1		
12				P4 = P1 to P2 by b*
13				P6 = P3 to P4 by a*
14				V = P5 to P6 by L*

As shown in Table 43, a single ADR register and adder can be used for address generation into the lookup tables. 6 sets of 8-bit registers can be used to hold intermediate results - 2 registers hold values loaded from the lookup tables, and 4 registers are used for the output from the interpolation unit. Note that the input to the linear interpolation unit is always a pair of 8-bit registers P1/P2, P3/P4, and P5/P6. This is done deliberately to reduce register selection logic. In cycle 14, the "V" register holds the 8-bit value finally calculated. The 8-bit result can be written to the appropriate location in Buffer 2 during the next cycle.

A block diagram of the Convert to CMY process can be seen in Figure 111.

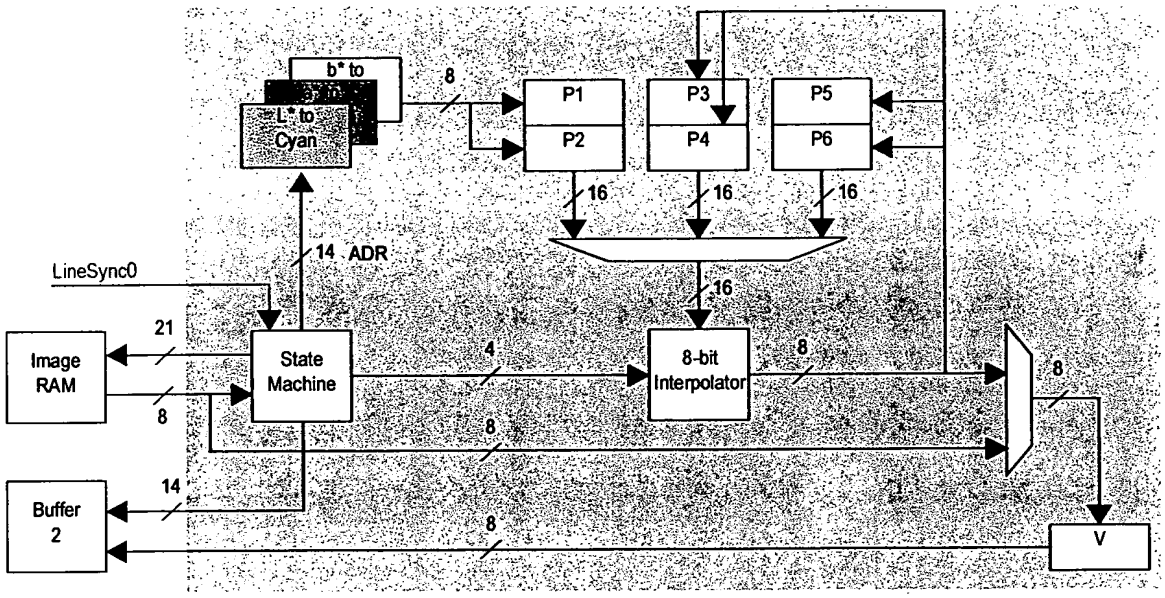


Figure 57. Convert from L*a*b* to CMY

Assuming the process is first run to generate cyan, the resultant cyan contone pixel is stored into the cyan contone buffer within Buffer 2. The process is then run again on the same L*a*b* input to generate the magenta pixel. This magenta contone pixel is stored into the magenta contone buffer of Buffer 2. Finally, the yellow contone pixel is generated from the same L*a*b* input, and the resultant yellow pixel is stored into the yellow contone buffer of Buffer 2).

The address generation for writing to the contone buffer (Buffer 2) is straightforward. A single address (and accompanying ColorSelect bits) is used to write to each of the three color buffers. The Cyan buffer is written to on cycle 15, the Magenta on cycle 30, and Yellow on cycle 45. The pixel address is incremented by 1 every 75 cycles (after all 3 colors

have been written). The line being written to increments with wrapping once every 6 LineSync0 pulses. The order of lines being written to is simply 0-1-2-3-4-5-0-1-2-3 etc...

If there is no conversion taking place (i.e. the image in ImageRAM is already in CMY format), then the address generation for Buffer 2 remains the same. The only difference is that the multiplexor chooses the value directly from Image RAM instead of from the result of the interpolator.

Although each line is 534 contone pixels, we only require 3200 bi-level dots. The scaling up by 6 by the up-interpolator gives 3204 dots. The up-interpolator simply stops after generating 3200 dots, and does not read the final contone pixel more than twice. Thus the writes ($36 \times 534 \times 3$) balance out with the reads ($9600 \times 6 - 4$).

Address generation for the Image RAM is very simple given that the image is stored in an interleaved fashion. A single address register contains the current address in ImageRAM. During cycles 1, 2, and 3 of the 14 cycle group the address is read and incremented, thus reading L*, a*, and b*, or C, M, Y. This is done 534 times and then the address generator stalls, waiting for the LineSync0 to enable the process to start again. The current address therefore gradually progresses through the entire image.

18.5.3.5 UpInterpolate, Halftone, and Reformat For Printer

Although the Up-Interpolate, Halftone, and Reformat For Printer tasks are defined as separate tasks by Section 17.2.2 on page 59, Section 17.2.3 on page 60, and Section 17.2.4 on page 61 respectively, they are implemented as a single process in the hardware implementation of the PPCP.

The input to the Up-interpolate, Halftone and Reformat Unit (UHRU) is the contone buffer (Buffer 2) containing the partial CMY image. The output is a set of 12-bit values in the correct order to be sent to the Memjet Interface for subsequent output to the printhead via Buffer 1. The 12 output bits are generated 1 bit at a time, and sent to the 12-bit shift register in Buffer 1.

The control of this process occurs from the Advance signal from the MJI and the LineSync0 pulse from the LSGU. When the UHRU starts up, and after each LineSync0 pulse, 12 bits are produced, and are clocked into the 12-bit shift register of Buffer 1. After the 12th bit has been clocked in, a TransferWriteEnable pulse is given to Buffer 1 and the next 12 bits are generated. After this, the UHRU waits for the Advance pulse from the MJI. When the Advance pulse arrives, the TransferWriteEnable pulse is given to Buffer 1, and the next 12 bits are calculated before waiting again. In practice, once the first Advance pulse is given, synchronization has occurred and future Advance pulses will occur every 12 cycles thereafter.

The UpInterpolate, Halftone and Reformat process can be seen in Figure 58.

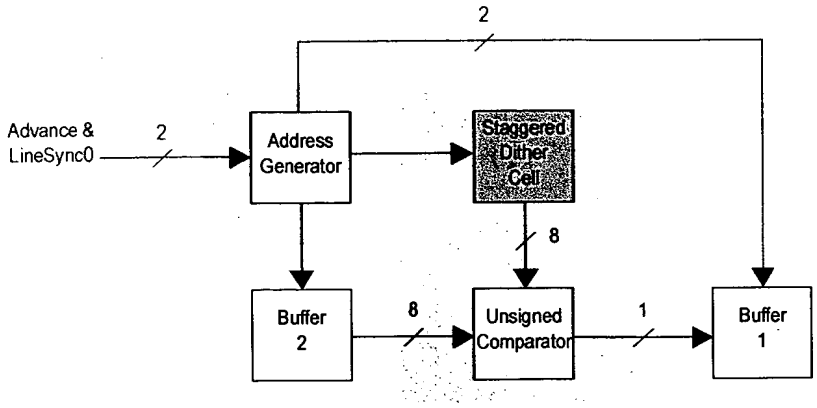


Figure 58. The UpInterpolate, Halftone and Reformat Process

The Halftone task is undertaken by the simple 8-bit unsigned comparator. The two inputs to the comparator come from the Staggered Dither Cell and Buffer 4. The order that these values are presented to the Unsigned Comparator is determined by the Address Generator State Machine, which ensures that the addresses into the CMY image stored in Buffer 2 match the segment-oriented order required for the printhead. The Address Generator State Machine therefore undertakes the Up-Interpolation and Reformatting for Printer tasks. Rather than simply access an entire line at a time at high resolution, and then reformat the line according to the printer lookup requirements (as described in Section 17.2.4 on page 61), the reformatting is achieved by the appropriate addressing of the CMY contone buffer (Buffer 2), and ensuring that the comparator uses the correct lookup from the dither cell to match the staggered addresses.

The Halftoning task is the same as described by Section 17.2.3 on page 60. However, since the dot outputs are generated in the correct order for the printhead, the size of the Dither Cell is chosen so that it divides evenly into 800. Consequently a given position in the dither cell for one segment will be the same for the remaining 3 segments. A 50×50 dither cell provides a satisfactory result. As described in Section 17.2.3 on page 60, the same position in the dither cell can be used for different colors due to the fact that different lines are being generated at the same time for each of the colors. The addressing for the dither cell is therefore quite simple. We start at a particular row in the Staggered Dither cell (e.g. row 0). The first dither cell entry used is Entry 0. We use that entry 12 times (12 cycles) to generate the 3 colors for all 4 segments, and then advance to Entry 1 of row 0. After Entry 49, we revert back to Entry 0. This continues for all 9,600 cycles in order to generate all 9,600 dots. The Halftone Unit then stops and waits for the LineSync0 pulse which causes the address generator to advance to the next row in the dither cell.

The Staggered Dither cell is so called because it differs from a regular dither cell by having the odd and even lines staggered. This is because we generate odd and even pixels (starting from pixel 0) on different lines, and saves the Address Generator from having to advance to the next row and back again on alternative sets of 12 pixels. Figure 59 shows a simple dither cell, and how to map it to a staggered dither cell of the same size. Note that

for determining the "oddness" of a given position, we number the pixels in a given row 0, 1, 2 etc.

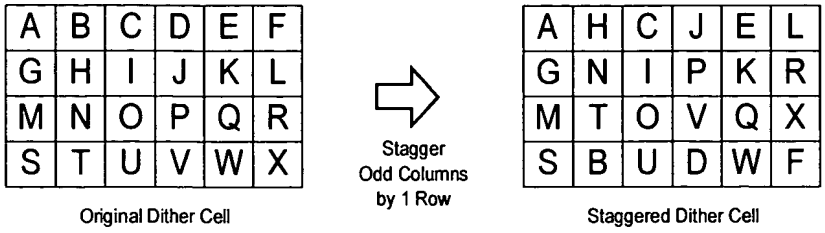


Figure 59. Mapping a Standard Dither Cell to the Staggered Pixel Dither Cell

The 8-bit value from Buffer 2 is compared (unsigned) to the 8-bit value from the Staggered Dither Cell. If the Buffer 2 pixel value is greater than or equal to the dither cell value, a "1" bit is output to the shift register of Buffer 1. Otherwise a "0" bit is output to the shift register of Buffer 1.

In order to halftone 9,600 contone pixels, 9,600 contone pixels must be read in. The Address Generator Unit performs this task, generating the addresses into Buffer 2, effectively implementing the UpInterpolate task. The address generation for reading Buffer 2 is slightly more complicated than the address generation for the dither cell, but not overly so.

The Address Generator for reading Buffer 2 only begins once the first row of Buffer 2 has been written. The remaining rows of Buffer 2 are 0, so they will effectively be white (no printed dots).

Each of the 6 effective output lines has a register with an integer and fractional component. The integer portion of the register is used to select which Buffer line will be read to effectively up-interpolate the color for that particular color's odd and even pixels. 3 pixel counters are used to maintain the current position within segment 0, and a single temporary counter P_ADR (pixel address) is used to offset into the remaining 3 segments.

In summary then, address generation for reading Buffer 2 requires the following registers, as shown in Table 20.

Table 20. Registers Required for Reading Buffer 2

Register Name	Size
CyanEven	6 bits (3:3)
CyanOdd	6 bits (3:3)
MagentaEven	6 bits (3:3)
MagentaOdd	6 bits (3:3)
YellowEven	6 bits (3:3)
YellowOdd	6 bits (3:3)
Cyan_P_ADR	13 bits (10:3)
Magenta_P_ADR	13 bits (10:3)
Yellow_P_ADR	13 bits (10:3)
P_ADR	13 bits (100:3)

The initial values for the 6 buffer line registers is the physical dot distance between nozzles (remember that the fractional component is effectively a divide by 6). For example, if the odd and even output dots of a color are separated by a distance of 1 dot, and nozzles of one color are separated from the nozzles of the next by 8 dots, the initial values would be as shown in First Line column in Table 21. Once each set of 9,600 dots has been generated, each of these counters must increment by 1 fractional component, representing the fact that we are sampling each pixel 6 times in the vertical dimension. The resultant values will then be as shown in Second Line column in Table 21. Note that 5:5 + 1 = 0:0 since there are only 6 buffer lines.

Table 21. Example Inital Setup and Second Line Values for the 6 Buffer Line Registers

Name	Calculation	First Line		Second Line	
		Value	Buff	Value	Buff
CyanEven	Initial Position	0:0	0	0:1	0
CyanOdd	CyanEven+0:1	0:1	0	0:2	0
MagentaEven	CyanOdd+1:2 (8)	1:3	1	1:4	1
MagentaOdd	MagentaEven+0:1	1:4	1	1:5	1
YellowEven	MagentaOdd+1:2 (8)	3:0	2	3:1	3
YellowOdd	YellowEven+0:1	3:1	3	3:2	3

The 6 buffer line registers then, determine which of the buffer lines is to be read for a given color's odd or even pixels. To determine which of the contone pixels are read from the specific line of Buffer 2, we use 3 Pixel Address counters, one for each color, and a single temporary counter (P_ADR) which is used to index into each segment. Each segment is separated from the next by 800 dots. In contone pixels this distance is 133:2 (800/6). We generate the 4 addresses for the even cyan pixels, then the 4 addresses for the even magenta, and finally the 4 addresses for the even yellow. We then do the same for the odd cyan, magenta, and yellow pixels. This process of two sets of 12 bits - 12 even then 12 odd, is performed 400 times. We can then reset the Pixel Address counters (X_P_ADR) to 0 and advance the 6 buffer line registers. Every 6 line advances, the next buffer line is now free and ready for updating (by the Convert to CMY process). Table 22 lists the steps in a simple form.

Table 22. Address Generation for Reading Buffer 2

#	Address	Calculation	Comment
0		P_ADR = Cyan_P_ADR Cyan_P_ADR += 0:1	Generate address for even pixel in Cyan segment 0 and advance to next pixel for cyan
1	CyanEven:P_ADR	P_ADR += 133:2	Advance to segment 1 (cyan)
2	CyanEven:P_ADR	P_ADR += 133:2	Advance to segment 2 (cyan)
3	CyanEven:P_ADR	P_ADR += 133:2	Advance to segment 3 (cyan)
4	CyanEven:P_ADR	P_ADR = Magenta_P_ADR Magenta_P_ADR += 0:1	Generate address for even pixel in Magenta segment 0 and advance to next pixel for magenta
5	MagentaEven:P_ADR	P_ADR += 133:2	Advance to segment 1 (magenta)
6	MagentaEven:P_ADR	P_ADR += 133:2	Advance to segment 2 (magenta)
7	MagentaEven:P_ADR	P_ADR += 133:2	Advance to segment 3 (magenta)

Table 22. Address Generation for Reading Buffer 2

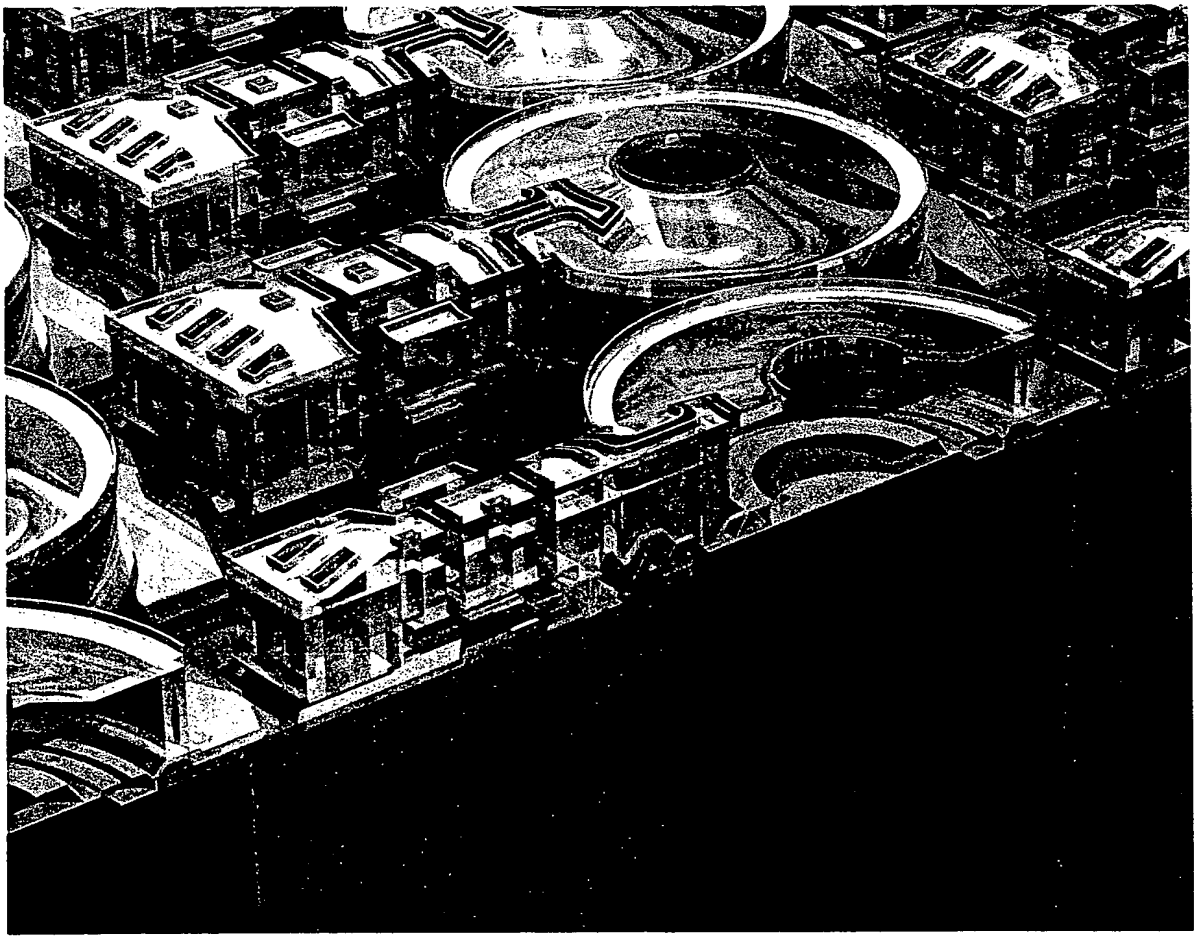
#	Address	Calculation	Comment
8	MagentaEven:P_ADR	P_ADR = Yellow_P_ADR Yellow_P_ADR += 0:1	Generate address for even pixel in Yellow segment 0 and advance to next pixel for yellow
9	YellowEven:P_ADR	P_ADR += 133:2	Advance to segment 1 (yellow)
10	YellowEven:P_ADR	P_ADR += 133:2	Advance to segment 2 (yellow)
11	YellowEven:P_ADR	P_ADR += 133:2	Advance to segment 3 (yellow)
12	YellowEven:P_ADR	P_ADR = Cyan_P_ADR Cyan_P_ADR += 0:1	Generate address for odd pixel in Cyan segment 0 and advance to next pixel for cyan
13	CyanOdd:P_ADR	P_ADR += 133:2	Advance to segment 1 (cyan)
etc.			

The pseudocode for generating the Buffer 2 addresses is shown here. Note that it is listed as a sequential set of steps. Table 22 shows a better view of the parallel nature of the operations during the address generation.

```
% Calculate start positions
CyanEven = 0:0
CyanOdd = CyanEven + 0:1
MagentaEven = CyanOdd + 1:2
MagentaOdd = MagentaEven + 0:1
YellowEven = MagentaOdd + 1:2
YellowOdd = YellowEven + 0:1

Do 5100 times (number of print lines)
   Cyan_P_ADR = 0:0
   Magenta_P_ADR = 0:0
   Yellow_P_ADR = 0:0
   Do 400 times
      % generate the even pixels for the first set of 12 bits
      P_ADR = Cyan_P_ADR
      Cyan_P_ADR += 0:1
      Do 4 times
         ReadBuffer2(line=CyanEven, pixel=P_ADR)
         P_ADR += 133:2
      EndDo
      P_ADR = Magenta_P_ADR
      Magenta_P_Adr += 0:1
      Do 4 times
         ReadBuffer2(line=MagentaEven, pixel=P_ADR)
         P_ADR += 133:2
      EndDo
      P_ADR = Yellow_P_ADR
      Yellow_P_Adr += 0:1
      Do 4 times
         ReadBuffer4(line=YellowEven, pixel=P_ADR)
         P_ADR += 133:2
      EndDo

      % generate the odd pixels for the first set of 12 bits
      P_ADR = Cyan_P_ADR
      Cyan_P_ADR += 0:1
      Do 4 times
         ReadBuffer2(line=CyanOdd, pixel=P_ADR)
         P_ADR += 133:2
```

```
      EndDo
      P_ADR = Magenta_P_ADR
      Magenta_P_Adr += 0:1
      Do 4 times
         ReadBuffer2(line=MagentaOdd, pixel=P_ADR)
         P_ADR += 133:2
      EndDo
      P_ADR = Yellow_P_ADR
      Yellow_P_Adr += 0:1
      Do 4 times
         ReadBuffer2(line=YellowOdd, pixel=P_ADR)
         P_ADR += 133:2
      EndDo

      % Now can advance to next "line"
      CyanEven += 0:1
      CyanOdd += 0:1
      MagentaEven += 0:1
      MagentaOdd += 0:1
      YellowEven += 0:1
      YellowOdd += 0:1
   EndDo
EndDo
```

Memjet Printhead



19 Introduction

A Memjet printhead is a drop-on-demand 1600 dpi inkjet printer that produces bi-level dots in up to 4 colors to produce a printed page of a particular width. Since the printhead prints dots at 1600 dpi, each dot is approximately 22.5μm in diameter, and spaced 15.875μm apart. Because the printing is bi-level, the input image should be dithered or error-diffused for best results.

Typically a Memjet printhead for a particular application is page-width. This enables the printhead to be stationary and allows the paper to move past the printhead. Figure 60 illustrates a typical configuration.

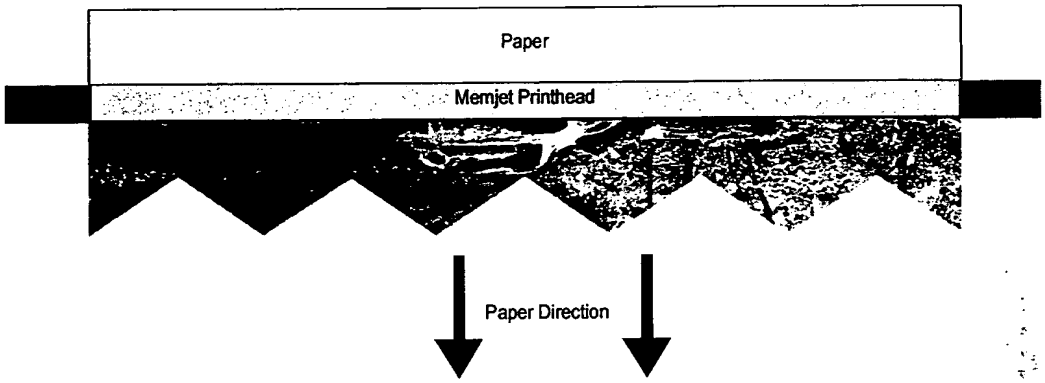


Figure 60. A Memjet printhead

A Memjet printhead is composed of a number of identical 1/2 inch Memjet segments. The segment is therefore the basic building block for constructing a printhead.

20 Memjet Segment Structure

This section examines the structure of a single segment, the basic building block for constructing Memjet printheads.

20.1 Grouping of Nozzles Within a Segment

The nozzles within a single segment are grouped for reasons of physical stability as well as minimization of power consumption during printing. In terms of physical stability, a total of 10 nozzles share the same ink reservoir. In terms of power consumption, groupings are made to enable a low-speed and a high-speed printing mode.

Memjet segments support two printing speeds to allow speed/power consumption tradeoffs to be made in different product configurations.

In the low-speed printing mode, 4 nozzles of each color are fired from the segment at a time. The exact number of nozzles fired depends on how many colors are present in the printhead. In a four color (e.g. CMYK) printing environment this equates to 16 nozzles firing simultaneously. In a three color (e.g. CMY) printing environment this equates to 12 nozzles firing simultaneously. To fire all the nozzles in a segment, 200 different sets of nozzles must be fired.

In the high-speed printing mode, 8 nozzles of each color are fired from the segment at a time. The exact number of nozzles fired depends on how many colors are present in the printhead. In a four color (e.g. CMYK) printing environment this equates to 32 nozzles firing simultaneously. In a three color (e.g. CMY) printing environment this equates to 24 nozzles firing simultaneously. To fire all the nozzles in a segment, 100 different sets of nozzles must be fired.

The power consumption in the low-speed mode is half that of the high-speed mode. Note, however, that the energy consumed to print a page is the same in both cases.

20.1.1 10 Nozzles Make a Pod

A single pod consists of 10 nozzles sharing a common ink reservoir. 5 nozzles are in one row, and 5 are in another. Each nozzle produces dots 22.5µm in diameter spaced on a 15.875µm grid to print at 1600 dpi. Figure 61 shows the arrangement of a single pod, with the nozzles numbered according to the order in which they must be fired.

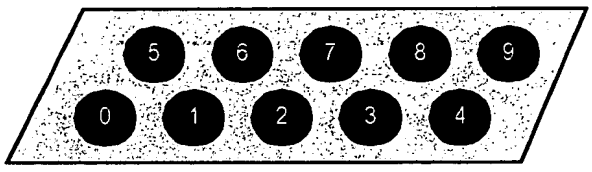


Figure 61. A single pod, numbered by firing order

Although the nozzles are fired in this order, the relationship of nozzles and physical placement of dots on the printed page is different. The nozzles from one row represent the even dots from one line on the page, and the nozzles on the other row represent the odd dots

from the adjacent line on the page. Figure 62 shows the same pod with the nozzles numbered according to the order in which they must be loaded.

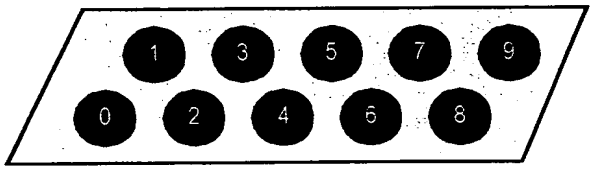


Figure 62. A single pod, numbered by load order

The nozzles within a pod are therefore logically separated by the width of 1 dot. The exact distance between the nozzles will depend on the properties of the Memjet firing mechanism. The printhead is designed with staggered nozzles designed to match the flow of paper.

20.1.2 1 Pod of Each Color Makes a Chromapod

One pod of each color are grouped together into a ***chromapod***. The number of pods in a chromapod will depend on the particular application. In a monochrome printing system (such as one that prints only black), there is only a single color and hence a single pod. Photo printing application printheads require 3 colors (cyan, magenta, yellow), so Memjet segments used for these applications will have 3 pods per chromapod (one pod of each color). The expected maximum number of pods in a chromapod is 4, as used in a CMYK (cyan, magenta, yellow, black) printing system (such as a desktop printer). This maximum of 4 colors is not imposed by any physical constraints - it is merely an expected maximum from the expected applications (of course, as the number of colors increases the cost of the segment increases and the number of these larger segments that can be produced from a single silicon wafer decreases).

A chromapod represents different color components of the same horizontal set of 10 dots on different lines. The exact distance between different color pods depends on the Memjet operating parameters, and may vary from one Memjet design to another. The distance is considered to be a constant number of dot-widths, and must therefore be taken into account when printing: the dots printed by the cyan nozzles will be for different lines than those printed by the magenta, yellow or black nozzles. The printing algorithm must allow for a variable distance up to about 8 dot-widths between colors. Figure 63 illustrates a single chromapod for a CMYK printing application.

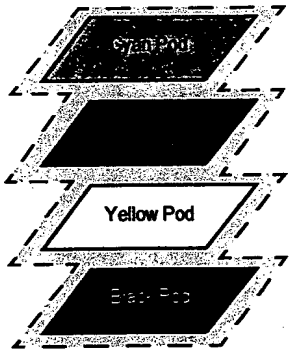


Figure 63. A Single Chromapod Contains 1 Pod of each Color

20.1.3 5 Chromapods Make a Podgroup

5 chromapods are organized into a single ***podgroup***. A podgroup therefore contains 50 nozzles for each color. The arrangement is shown in Figure 64, with chromapods numbered 0-4 and using a CMYK chromapod as the example. Note that the distance between adjacent chromapods is exaggerated for clarity.

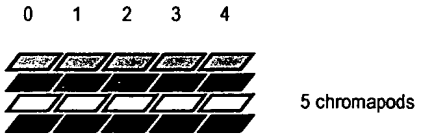


Figure 64. A Single Podgroup Contains 5 Chromapods

20.1.4 2 Podgroups Make a Phasegroup

2 podgroups are organized into a single ***phasegroup***. The phasegroup is so named because groups of nozzles within a phasegroup are fired simultaneously during a given firing phase (this is explained in more detail below). The formation of a phasegroup from 2 podgroups is entirely for the purposes of low-speed and high-speed printing via 2 PodgroupEnable lines.

During low-speed printing, only one of the two PodgroupEnable lines is set in a given firing pulse, so only one podgroup of the two fires nozzles. During high-speed printing, both PodgroupEnable lines are set, so both podgroups fire nozzles. Consequently a low-speed print takes twice as long as a high-speed print, since the high-speed print fires twice as many nozzles at once.

Figure 65 illustrates the composition of a phasegroup. The distance between adjacent podgroups is exaggerated for clarity.

Figure 65. A single phasegroup contains 2 podgroups

20.1.5 2 Phasegroups Make a Firegroup

Two phasegroups (PhasegroupA and PhasegroupB) are organized into a single ***firegroup***, with 4 firegroups in each segment. Firegroups are so named because they all fire the same nozzles simultaneously. Two enable lines, AEnable and BEnable, allow the firing of PhasegroupA nozzles and PhasegroupB nozzles independently as different firing phases. The

arrangement is shown in Figure 66. The distance between adjacent groupings is exaggerated for clarity.

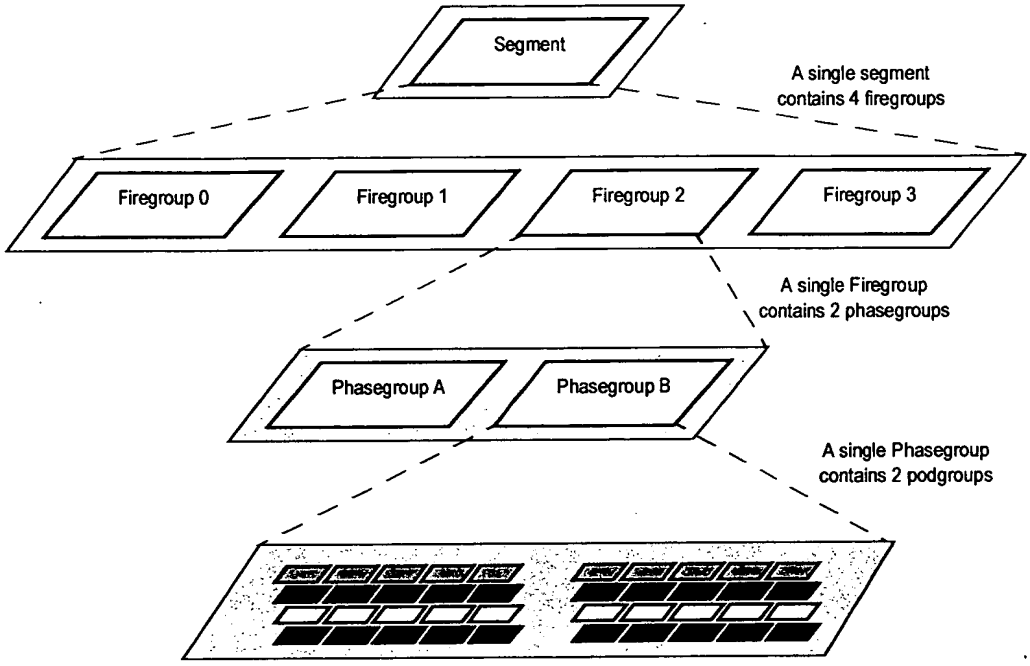


Figure 66. Relationship Between Segments, Firegroups, Phasegroups, Podgroups and ChromaPods

20.1.6 Nozzle Grouping Summary

Table 23 is a summary of the nozzle groupings in a segment assuming a CMYK chromapod.

Table 23. Nozzle Groupings for a single segment

Name of Grouping	Composition	Replication Ratio	Nozzle Count
Nozzle	Base unit	1:1	1
Pod	Nozzles per pod	10:1	10
Chromapod	Pods per chromapod	C:1	10C
Podgroup	Chromapods per podgroup	5:1	50C
Phasegroup	Podgroups per phasegroup	2:1	100C
Firegroup	Phasegroups per firegroup	2:1	200C
Segment	Firegroups per segment	4:1	800C

The value of C, the number of colors contained in the segment, determines the total number of nozzles.

- With a 4 color segment, such as CMYK, the number of nozzles per segment is 3,200.
- With a 3 color segment, such as CMY, the number of nozzles per segment is 2,400.
- In a monochrome environment, the number of nozzles per segment is 800.

20.2 LOAD AND PRINT CYCLES

A single segment contains a total of 800C nozzles, where C is the number of colors in the segment. A *Print Cycle* involves the firing of up to all of these nozzles, dependent on the information to be printed. A *Load Cycle* involves the loading up of the segment with the information to be printed during the subsequent Print Cycle.

Each nozzle has an associated ***NozzleEnable*** bit that determines whether or not the nozzle will fire during the Print Cycle. The NozzleEnable bits (one per nozzle) are loaded via a set of shift registers.

Logically there are C shift registers per segment (one per color), each 800 deep. As bits are shifted into the shift register for a given color they are directed to the lower and upper nozzles on alternate pulses. Internally, each 800-deep shift register is comprised of two 400-deep shift registers: one for the upper nozzles, and one for the lower nozzles. Alternate bits are shifted into the alternate internal registers. As far as the external interface is concerned however, there is a single 800 deep shift register.

Once all the shift registers have been fully loaded (800 load pulses), all of the bits are transferred in parallel to the appropriate NozzleEnable bits. This equates to a single parallel transfer of 800C bits. Once the transfer has taken place, the Print Cycle can begin. The Print Cycle and the Load Cycle can occur simultaneously as long as the parallel load of all NozzleEnable bits occurs at the end of the Print Cycle.

20.2.1 Load Cycle

The Load Cycle is concerned with loading the segment's shift registers with the next Print Cycle's NozzleEnable bits.

Each segment has C inputs directly related to the C shift registers (where C is the number of colors in the segment). These inputs are named ***ColorNData***, where N is 1 to C (for example, a 4 color segment would have 4 inputs labeled ***Color1Data***, ***Color2Data***, ***Color3Data*** and ***Color4Data***). A single pulse on the ***SRClock*** line transfers C bits into the appropriate shift registers. Alternate pulses transfer bits to the lower and upper nozzles respectively. A total of 800 pulses are required for the complete transfer of data. Once all 800C bits have been transferred, a single pulse on the ***PTransfer*** line causes the parallel transfer of data from the shift registers to the appropriate NozzleEnable bits.

The parallel transfer via a pulse on PTransfer must take place *after* the Print Cycle has finished. Otherwise the NozzleEnable bits for the line being printed will be incorrect.

It is important to note that the odd and even dot outputs, although printed during the same Print Cycle, do not appear on the same physical output line. The physical separation of odd and even nozzles within the printhead, as well as separation between nozzles of different colors ensures that they will produce dots on different lines of the page. This relative difference must be accounted for when loading the data into the printhead. The actual difference in lines depends on the characteristics of the inkjet mechanism used in the printhead. The differences can be defined by variables D_1 and D_2 where D_1 is the distance

between nozzles of different colors, and D_2 is the distance between nozzles of the same color. Table 24 shows the dots transferred to a C color segment on the first 4 pulses.

Table 24. Order of Dots Transferred to a Segment

Pulse	Dot	Color1 Line	Color2 Line	Color3 Line	ColorC Line
1	0	N	$N+D_1$[a]	$N+2D_1$	$N+(C-1)D_1$
2	1	$N+D_2$[b]	$N+D_1+D_2$	$N+2D_1+D_2$	$N+(C-1)D_1+D_2$
3	2	N	$N+D_1$	$N+2D_1$	$N+(C-1)D_1$
4	3	$N+D_2$	$N+D_1+D_2$	$N+2D_1+D_2$	$N+(C-1)D_1+D_2$

a. D_1 = number of lines between the nozzles of one color and the next (likely = 4-8)

b. D_2 = number of lines between two rows of nozzles of the same color (likely = 1)

And so on for all 800 pulses.

Data can be clocked into a segment at a maximum rate of 20 MHz, which will load the entire 800C bits of data in 40µs.

20.2.2 Print Cycle

A single Memjet printhead segment contains 800 nozzles. To fire them all at once would consume too much power and be problematic in terms of ink refill and nozzle interference. This problem is made more apparent when we consider that a Memjet printhead is composed of multiple 1/2 inch segments, each with 800 nozzles. Consequently two firing modes are defined: a low-speed printing mode and a high-speed printing mode:

- In the low-speed print mode, there are 200 phases, with each phase firing 4C nozzles (C per firegroup, where C is the number of colors).
- In the high-speed print mode, there are 100 phases, with each phase firing 8C nozzles, (2C per firegroup, where C is the number of colors).

The nozzles to be fired in a given firing pulse are determined by

- 3 bits ***ChromapodSelect*** (select 1 of 5 chromapods from a firegroup)
- 4 bits ***NozzleSelect*** (select 1 of 10 nozzles from a pod)
- 2 bits of ***PodgroupEnable*** lines (select 0, 1, or 2 podgroups to fire)

When one of the PodgroupEnable lines is set, only the specified Podgroup's 4 nozzles will fire as determined by ChromapodSelect and NozzleSelect. When both of the PodgroupEnable lines are set, both of the podgroups will fire their nozzles. For the low-speed mode, two fire pulses are required, with PodgroupEnable = 10 and 01 respectively. For the high-speed mode, only one fire pulse is required, with PodgroupEnable = 11.

The duration of the firing pulse is given by the ***AEnable*** and ***BEnable*** lines, which fire the PhasegroupA and PhasegroupB nozzles from all firegroups respectively. The typical duration of a firing pulse is 1.3 - 1.8 µs. The duration of a pulse depends on the viscosity of the ink (dependent on temperature and ink characteristics) and the amount of power available to the printhead. See Section 20.3 on page 97 for details on feedback from the printhead in order to compensate for temperature change.

The AEnable and BEnable are separate lines in order that the firing pulses can overlap. Thus the 200 phases of a low-speed Print Cycle consist of 100 A phases and 100 B phases,

effectively giving 100 sets of Phase A and Phase B. Likewise, the 100 phases of a high-speed print cycle consist of 50 A phases and 50 B phases, effectively giving 50 phases of phase A and phase B.

Figure 67 shows the AEnable and BEnable lines during a typical Print Cycle. In a high-speed print there are 50 2µs cycles, while in a low-speed print there are 100 2µs cycles.

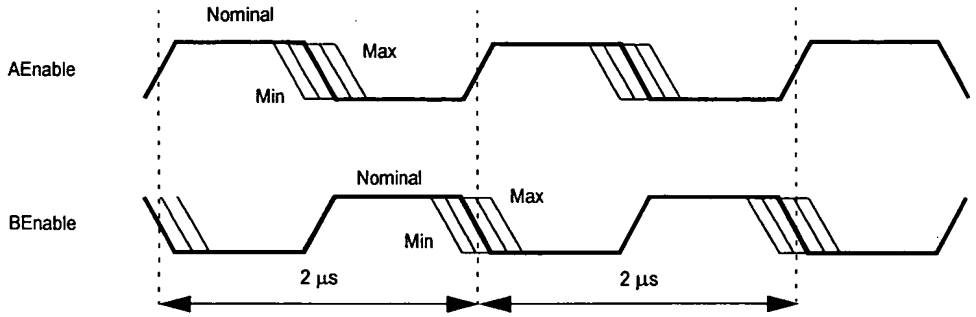


Figure 67. AEnable and BEnable During a Typical Print Cycle

For the high-speed printing mode, the firing order is:

- ChromapodSelect 0, NozzleSelect 0, PodgroupEnable 11 (Phases A and B)
- ChromapodSelect 1, NozzleSelect 0, PodgroupEnable 11 (Phases A and B)
- ChromapodSelect 2, NozzleSelect 0, PodgroupEnable 11 (Phases A and B)
- ChromapodSelect 3, NozzleSelect 0, PodgroupEnable 11 (Phases A and B)
- ChromapodSelect 4, NozzleSelect 0, PodgroupEnable 11 (Phases A and B)
- ChromapodSelect 0, NozzleSelect 1, PodgroupEnable 11 (Phases A and B)
- ...
- ChromapodSelect 3, NozzleSelect 9, PodgroupEnable 11 (Phases A and B)
- ChromapodSelect 4, NozzleSelect 9, PodgroupEnable 11 (Phases A and B)

For the low-speed printing mode, the firing order is similar. For each phase of the high speed mode where PodgroupEnable was 11, two phases of PodgroupEnable = 01 and 10 are substituted as follows:

- ChromapodSelect 0, NozzleSelect 0, PodgroupEnable 01 (Phases A and B)
- ChromapodSelect 0, NozzleSelect 0, PodgroupEnable 10 (Phases A and B)
- ChromapodSelect 1, NozzleSelect 0, PodgroupEnable 01 (Phases A and B)
- ChromapodSelect 1, NozzleSelect 0, PodgroupEnable 10 (Phases A and B)
- ...
- ChromapodSelect 3, NozzleSelect 9, PodgroupEnable 01 (Phases A and B)
- ChromapodSelect 3, NozzleSelect 9, PodgroupEnable 10 (Phases A and B)
- ChromapodSelect 4, NozzleSelect 9, PodgroupEnable 01 (Phases A and B)
- ChromapodSelect 4, NozzleSelect 9, PodgroupEnable 10 (Phases A and B)

When a nozzle fires, it takes approximately 100μs to refill. The nozzle cannot be fired before this refill time has elapsed. This limits the fastest printing speed to 100μs per line. In the high-speed print mode, the time to print a line is 100μs, so the time between firing a nozzle from one line to the next matches the refill time. The low-speed print mode is slower than this, so is also acceptable.

The firing of a nozzle also causes acoustic perturbations for a limited time within the common ink reservoir of that nozzle's pod. The perturbations can interfere with the firing of another nozzle within the same pod. Consequently, the firing of nozzles within a pod should be offset from each other as long as possible. We therefore fire four nozzles from a chromapod (one nozzle per color) and then move onto the next chromapod within the podgroup.

- In the low-speed printing mode the podgroups are fired separately. Thus the 5 chromapods within both podgroups must all fire before the first chromapod fires again, totalling 10 × 2μs cycles. Consequently each pod is fired once per 20μs.
- In the high-speed printing mode, the podgroups are fired together. Thus the 5 chromapods within a single podgroups must all fire before the first chromapod fires again, totalling 5 × 2μs cycles. Consequently each pod is fired once per 10 μs.

As the ink channel is 300μm long and the velocity of sound in the ink is around 1500m/s, the resonant frequency of the ink channel is 2.5MHz. Thus the low-speed mode allows 50 resonant cycles for the acoustic pulse to dampen, and the high-speed mode allows 25 resonant cycles. Consequently any acoustic interference is minimal in both cases.

20.3 FEEDBACK FROM A SEGMENT

A segment produces several lines of feedback. The feedback lines are used to adjust the timing of the firing pulses. Since multiple segments are collected together into a printhead, it is effective to share the feedback lines as a tri-state bus, with only one of the segments placing the feedback information on the feedback lines.

A pulse on the segment's ***SenseSegSelect*** line ANDed with data on ***Color1Data*** selects if the particular segment will provide the feedback. The feedback sense lines will come from that segment until the next SenseSegSelect pulse. The feedback sense lines are as follows:

- ***Tsense*** informs the controller how hot the printhead is. This allows the controller to adjust timing of firing pulses, since temperature affects the viscosity of the ink.
- ***Vsense*** informs the controller how much voltage is available to the actuator. This allows the controller to compensate for a flat battery or high voltage source by adjusting the pulse width.
- ***Rsense*** informs the controller of the resistivity (Ohms per square) of the actuator heater. This allows the controller to adjust the pulse widths to maintain a constant energy irrespective of the heater resistivity.
- ***Wsense*** informs the controller of the width of the critical part of the heater, which may vary up to ± 5% due to lithographic and etching variations. This allows the controller to adjust the pulse width appropriately.

20.4 PREHEAT CYCLE

The printing process has a strong tendency to stay at the equilibrium temperature. To ensure that the first section of the printed photograph has a consistent dot size, the equilibrium temperature must be met *before* printing any dots. This is accomplished via a preheat cycle.

The Preheat cycle involves a single Load Cycle to all nozzles of a segment with 1s (i.e. setting all nozzles to fire), and a number of short firing pulses to each nozzle. The duration of the pulse must be insufficient to fire the drops, but enough to heat up the ink. Altogether about 200 pulses for each nozzle are required, cycling through in the same sequence as a standard Print Cycle.

Feedback during the Preheat mode is provided by Tsense, and continues until equilibrium temperature is reached (about 30° C above ambient). The duration of the Preheat mode is around 50 milliseconds, and depends on the ink composition.

Preheat is performed before each print job. This does not affect performance as it is done while the data is being transferred to the printer.

20.5 CLEANING CYCLE

In order to reduce the chances of nozzles becoming clogged, a cleaning cycle can be undertaken before each print job. Each nozzle is fired a number of times into an absorbent sponge.

The cleaning cycle involves a single Load Cycle to all nozzles of a segment with 1s (i.e. setting all nozzles to fire), and a number of firing pulses to each nozzle. The nozzles are cleaned via the same nozzle firing sequence as a standard Print Cycle. The number of

times that each nozzle is fired depends upon the ink composition and the time that the printer has ben idle. As with preheat, the cleaning cycle has no effect on printer performance.

20.6 PRINTHEAD INTERFACE SUMMARY

Each segment has the following connections to the bond pads:

Table 25. Segment Interface Connections

Name	Lines	Description
Chromapod Select	3	Select which chromapod will fire (0-4)
NozzleSelect	4	Select which nozzle from the pod will fire (0-9)
PodgroupEnable	2	Enable the podgroups to fire (choice of: 01, 10, 11)
AEnable	1	Firing pulse for podgroup A
BEnable	1	Firing pulse for podgroup B
ColorNData	C	Input to shift registers (1 bit for each of C colors in the segment)
SRClock	1	A pulse on SRClock (ShiftRegisterClock) loads C bits from Color-Data into the C shift registers.
PTransfer	1	Parallel transfer of data from the shift registers to the internal NozzleEnable bits (one per nozzle).
SenseSegSelect	1	A pulse on SenseSegSelect ANDed with data on Color1Data selects the sense lines for this segment.
Tsense	1	Temperature sense
Vsense	1	Voltage sense
Rsense	1	Resistivity sense
Wsense	1	Width sense
Logic GND	1	Logic ground
Logic PWR	1	Logic power
V-	21	Actuator Ground
V+	21	Actuator Power
TOTAL	62+C	(if C is 4, Total = 66)

21 Making Memjet Printheads out of Segments

A Memjet printhead is composed of a number of identical 1/2 inch printhead segments. These 1/2 inch segments are manufactured together or placed together after manufacture to produce a printhead of the desired length. Each 1/2 inch segments prints 800 1600 dpi bi-level dots in up to 4 colors over a different part of the page to produce the final image. Although each segment produces 800 dots of the final image, each dot is represented by a combination of colored inks.

A 4-inch printhead, for example, consists of 8 segments, typically manufactured as a monolithic printhead. In a typical 4-color printing application (cyan, magenta, yellow, black), each of the segments prints bi-level cyan, magenta, yellow and black dots over a different part of the page to produce the final image. The positions of the segments are shown in Figure 68.

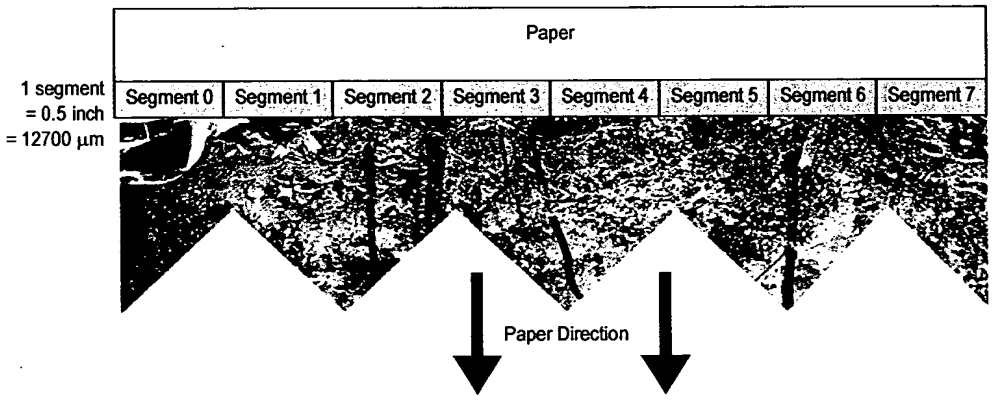


Figure 68. Arrangement of Segments in a 4-inch Printhead

An 8-inch printhead can be constructed from two 4-inch printheads or from a single 8-inch printhead consisting of 16 segments. Regardless of the construction mechanism, the effective printhead is still 8 inches in length.

A 2-inch printhead has a similar arrangement, but only uses 4 segments. Likewise, a full-bleed A4/Letter printer uses 17 segments for an effective 8.5 inch printing area.

Since the total number of nozzles in a segment is 800C (see Table 23), the total number of nozzles in a given printhead with S segments is 800CS. Thus segment N is responsible for printing dots 800N to 800N+799.

A number of considerations must be made when wiring up a printhead. As the width of the printhead increases, the number of segments increases, and the number of connections also increases. Each segment has its own ColorData connections (C of them), as well as SRClock and other connections for loading and printing.

21.1 LOADING CONSIDERATIONS

When the number of segments *S* is small it is reasonable to load all the segments simultaneously by using a common SRClock line and placing C bits of data on each of the ColorData inputs for the segments. In a 4-inch printer, S=8, and therefore the total number of bits to transfer to the printhead in a single SRClock pulse is 32. However for an 8-inch printer, S=16, and it is unlikely to be reasonable to have 64 data lines running from the print data generator to the printhead.

Instead, it is convenient to group a number of segments together for loading purposes. Each group of segments is small enough to be loaded simultaneously, and share an SRClock. For example, an 8-inch printhead can have 2 segment groups, each segment group containing 8 segments. 32 ColorData lines can be shared for both groups, with 2 SRClock lines, one per segment group.

When the number of segment groups is not easily divisible, it is still convenient to group the segments. One example is a 8.5 inch printer for producing A4/Letter pages. There are 17 segments, and these can be grouped as two groups of 9 (9C bits of data going to each segment, with all 9C bits used in the first group, and only 8C bits used for the second group), or as 3 groups of 6 (again, C bits are unused in the last group).

As the number of segment groups increases, the time taken to load the printhead increases. When there is only one group, 800 load pulses are required (each pulse transfers C data bits). When there are G groups, 800G load pulses are required. The bandwidth of the connection between the data generator and the printhead must be able to cope and be within the allowable timing parameters for the particular application.

If ***G*** is the number of segment groups, and ***L*** is the largest number of segments in a group, the printhead requires LC ColorData lines and G SRClock lines. Regardless of G, only a single PTransfer line is required - it can be shared across all segments.

Since L segments in each segment group are loaded with a single SRClock pulse, any printing process must produce the data in the correct sequence for the printhead. As an example, when G=2 and L=4, the first SRClock1 pulse will transfer the ColorData bits for the next Print Cycle's dot 0, 800, 1600, and 2400. The first SRClock2 pulse will transfer the ColorData bits for the next Print Cycle's dot 3200, 4000, 4800, and 5600. The second SRClock1 pulse will transfer the ColorData bits for the next Print Cycle's dot 1, 801, 1601, and 2401. The second SRClock2 pulse will transfer the ColorData bits for the next Print Cycle's dot 3201, 4001, 4801 and 5601.

After 800G SRClock pulses (800 to each of SRClock1 and SRClock2), the entire line has been loaded into the printhead, and the common PTransfer pulse can be given.

It is important to note that the odd and even color outputs, although printed during the same Print Cycle, do not appear on the same physical output line. The physical separation of odd and even nozzles within the printhead, as well as separation between nozzles of different colors ensures that they will produce dots on different lines of the page. This relative difference must be accounted for when loading the data into the printhead. The actual difference in lines depends on the characteristics of the inkjet mechanism used in the printhead. The differences can be defined by variables D_1 and D_2 where D_1 is the distance between nozzles of different colors, and D_2 is the distance between nozzles of the same

color. Considering only a single segment group, Table 26 shows the dots transferred to segment *n* of a printhead during the first 4 pulses of the shared SRClock.

Table 26. Order of Dots Transferred to a Segment in a Printhead

Pulse	Dot	Color1 Line	Color2 Line	ColorC Line
1	800S[a]	N	N+D_1[b]	N+(C-1)D_1
2	800S+1	N+D_2[c]	N+D_1+D_2	N+(C-1)D_1+D_2
3	800S+2	N	N+D_1	N+(C-1)D_1
4	800S+3	N+D_2	N+D_1+D_2	N+(C-1)D_1+D_2

a. S = segment number

b. D_1 = number of lines between the nozzles of one color and the next (likely = 4-8)

c. D_2 = number of lines between two rows of nozzles of the same color (likely = 1)

And so on for all 800 SRClock pulses to the particular segment group.

21.2 PRINTING CONSIDERATIONS

With regards to printing, we print 4C nozzles from each segment in the low-speed printing mode, and 8C nozzles from each segment in the high speed printing mode.

While it is certainly possible to wire up segments in any way, this document only considers the situation where all segments fire simultaneously. This is because the low-speed printing mode allows low-power printing for small printheads (e.g. 2-inch and 4-inch), and the controller chip design assumes there is sufficient power available for the large print sizes (such as 8-18 inches). It is a simple matter to alter the connections in the printhead to allow grouping of firing should a particular application require it.

When all segments are fired at the same time 4CS nozzles are fired in the low-speed printing mode and 8CS nozzles are fired in the high-speed printing mode. Since all segments print simultaneously, the printing logic is the same as defined in Section 20.2.2 on page 94.

The timing for the two printing modes is therefore:

- 200 µs to print a line at low speed (comprised of 100 2µs cycles)
- 100 µs to print a line at high speed (comprised of 50 2µs cycles)

21.3 FEEDBACK CONSIDERATIONS

A segment produces several lines of feedback, as defined in Section 20.3 on page 97. The feedback lines are used to adjust the timing of the firing pulses. Since multiple segments are collected together into a printhead, it is effective to share the feedback lines as a tri-state bus, with only one of the segments placing the feedback information on the feedback lines at a time.

Since the selection of which segment will place the feedback information on the shared Tsense, Vsense, Rsense, and Wsense lines uses the Color1Data line, the groupings of segments for loading data can be used for selecting the segment for feedback.

Just as there are G SRClock lines (a single line is shared between segments of the same segment group), there are G SenseSegSelect lines shared in the same way. When the correct SenseSegSelect line is pulsed, the segment of that group whose Color1Data bit is set will start to place data on the shared feedback lines. The segment previously active in terms of feedback must also be disabled by having a 0 on its Color1Data bit, and this segment may be in a different segment group. Therefore when there is more than one segment group. changing the feedback segment requires two steps: disabling the old segment, and enabling the new segment.

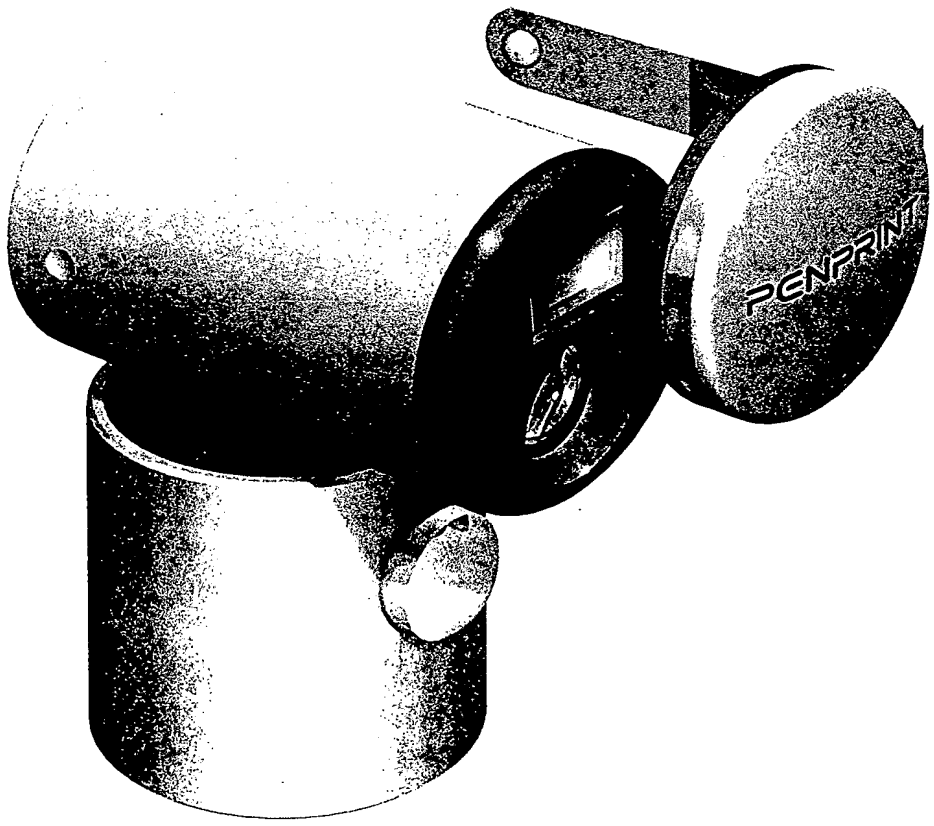
21.4 PRINTHEAD CONNECTION SUMMARY

This section assumes that a printhead has been constructed from a number of segments as described in the previous sections. It assumes that for data loading purposes, the segments have been grouped into G segment groups, with L segments in the largest segment group. It assumes there are C colors in the printhead. It assumes that the firing mechanism for the printhead is that all segments fire simultaneously, and only one segment at a time places feedback information on a common tri-state bus. Assuming all these things, Table 27 lists the external connections that are available from a printhead:

Table 27. Printhead Connections

Name	#Pins	Description
ChromapodSelect	3	Select which chromapod will fire (0-4)
NozzleSelect	4	Select which nozzle from the pod will fire (0-9)
PodgroupEnable	2	Enable the podgroups to fire (choice of: 01, 10, 11)
AEnable	1	Firing pulse for phasegroup A
BEnable	1	Firing pulse for phasegroup B
ColorData	CL	Inputs to C shift registers of segments 0 to L-1
SRClock	G	A pulse on SRClock[N] (ShiftRegisterClock N) loads the current values from ColorData lines into the L segments in segment group N.
PTransfer	1	Parallel transfer of data from the shift registers to the internal NozzleEnable bits (one per nozzle).
SenseSegSelect	G	A pulse on SenseSegSelect N ANDed with data on Color1Data[n] selects the sense lines for segment n in segment group N.
Tsense	1	Temperature sense
Vsense	1	Voltage sense
Rsense	1	Resistivity sense
Wsense	1	Width sense
Logic GND	1	Logic ground
Logic PWR	1	Logic power
V-	Bus bars	Actuator Ground
V+		Actuator Power
TOTAL	18+2G+CL	

Camera Module



22 Introduction

22.1 Overview

The PenPrint Camera Module provides a point-and-shoot camera component to a PenPrint system as a means of capturing images. The Camera Module is a standard PenPrint module containing an image sensor and specialized image processing chip. Captured images are transferred to the central PenPrint Printer Module for subsequent printing out, manipulation, or storage. The Camera Module also contains a self-timer mode similar to that found on regular cameras.

The Camera Module makes use of the Flash Module (see Section 9 on page 32) if it is detected in the current PenPrint configuration.

22.2 Appearance

Figure 69 shows a magnified illustration of the PenPrint Camera Module. Note the swivel connection for the lens assembly.

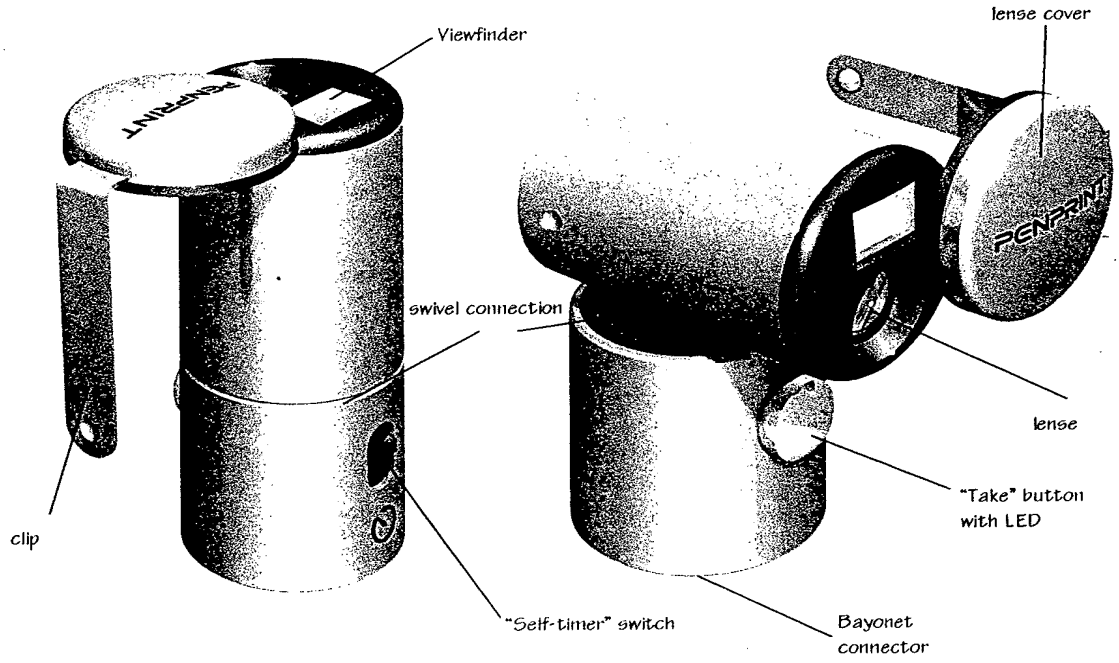


Figure 69. PenPrint Camera Module

22.3 Method of Operation

The Camera Module simply connects to a PenPrint configuration via the standard PenPrint bayonet connector. Power is provided from the PenPrint Printer module via the PenPrint Serial Bus.

To capture an image, a user simply presses the ***Take*** button. The viewfinder allows the user to frame the image before pressing the ***Take*** button. The swivel connection on the lens assembly assists in directing the lens to the correct orientation if the remainder of the PenPrint configuration is fixed.

When the ***Take*** button is pressed, the image is captured through the lens and transferred to the Printer Module. If the ***Take*** button is pressed again, a new image will be captured and transferred to the Printer Module. The image is always transferred to the Printer module once the Take button is pressed. Note that although the image remains in the Camera Module, there is no physical method of transferring the same image from the Camera Module again. The image must be saved from the Printer Module instead (to, for example, a Memory Module). The only way of directly accessing the captured image is via the computer interface (see Section 22.4 below). This limitation was chosen simply to reduce the usability complexity on the Camera Module.

The self-timer switch set to off/on disables and enables a 10 second delay between the pressing of the ***Take*** button and the capturing of the image. A LED inside the ***Take*** button provides a visual feedback during the countdown. The LED flashes once per second, and then stays on for the final two seconds of the countdown. The self-timing functionality is therefore identical to that of a conventional camera.

If there is an active Flash Module (see Section 9 on page 32) present in the PenPrint configuration, then the Flash will be activated depending on the Flash Module's flash mode. If the Flash Module has been turned ***off***, then the flash will not fire. If the Flash Module is set to ***auto***, then the flash fires as necessary (light detection carried out by the Camera Module). See Section 9 on page 32 for more information.

22.4 COMPUTER INTERFACE

The Camera Module can be instructed to take a photo either by a computer (via the USB Module) or by another module. However in both cases, the self-timer switch is ignored and the captured image is *not* transferred to the Printer Module. Instead, the image is simply captured and stored locally in the Camera Module. The Camera Module can then be instructed in a subsequent command to transfer its image to a specified module or simply to return it to the caller.

22.5 INTERNALS

The Camera Module consists of:

- standard PenPrint bayonet connector for easy attachment to a PenPrint configuration
- a simple lens assembly
- a CMOS imaging sensor
- a PenPrint Camera Image Processing chip (an ASIC designed to produce high quality images from the CMOS image sensor).

Partially and fully exploded views of the Camera Module internals are shown in Figure 70 and Figure 71.

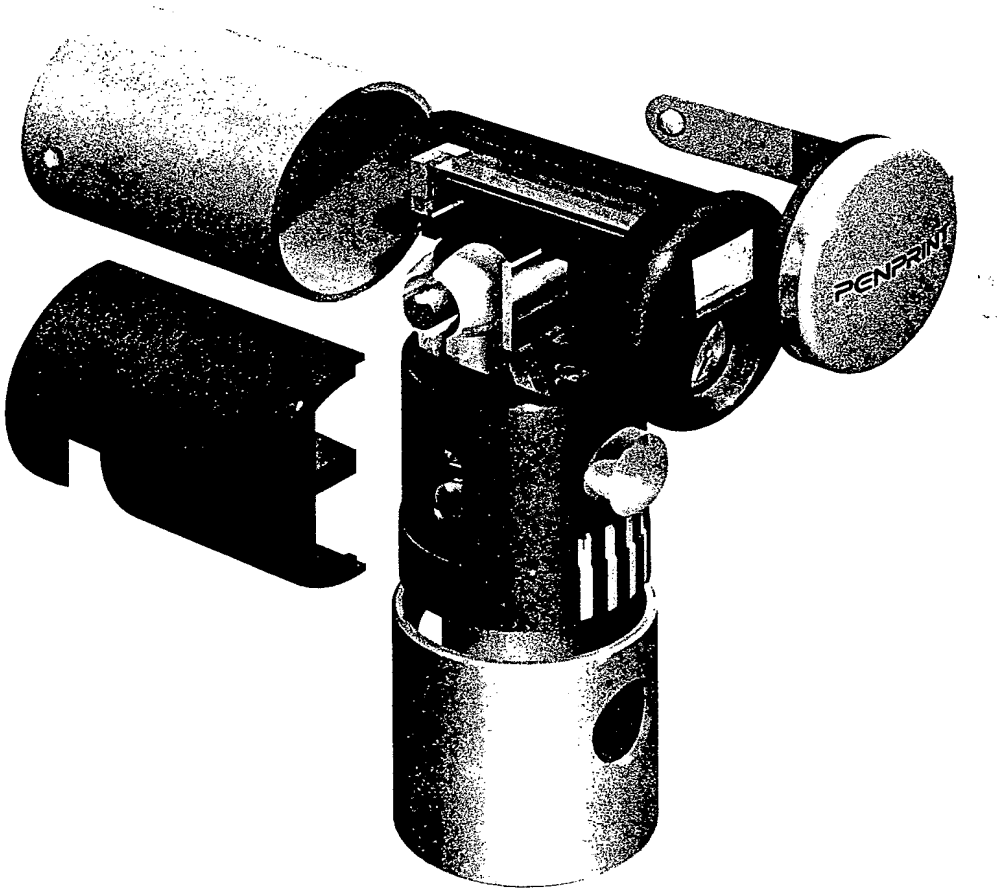


Figure 70. Partially Exploded View of Camera Module

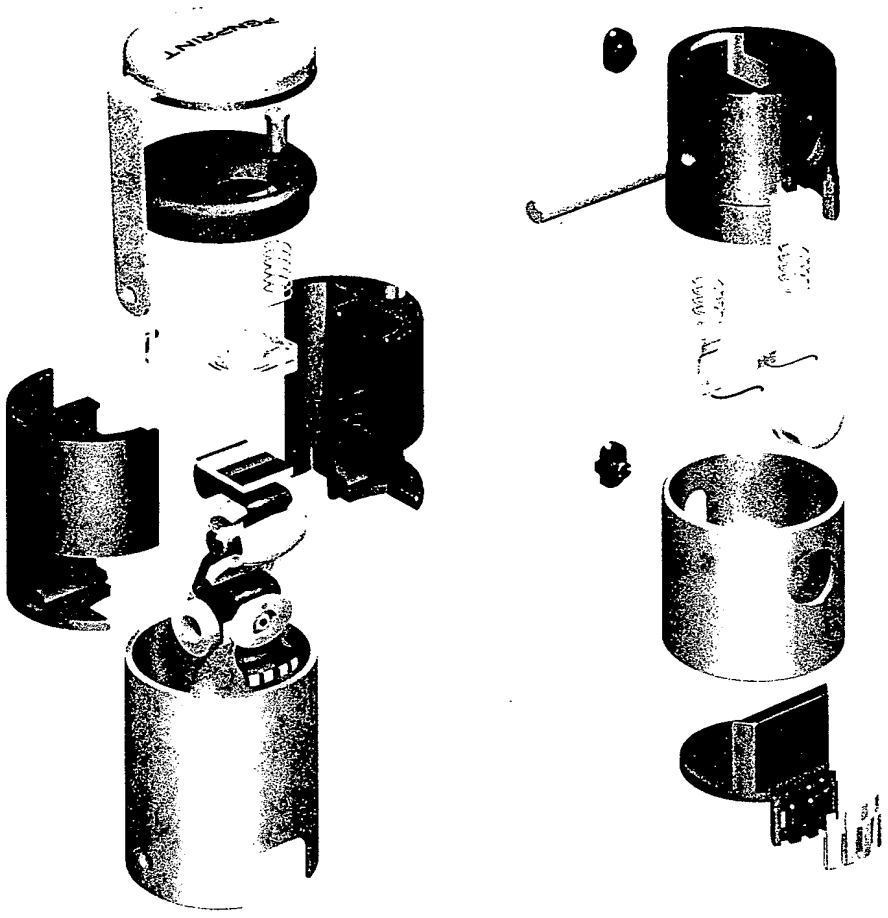


Figure 71. Fully Exploded View of Camera Module

23 Image Processing Requirements

There are a number of steps involved in taking an image from the image sensor, and producing a high quality output print. While the PenPrint Printer Module is responsible for producing the final output print, the job of the PenPrint Camera Module is to provide an image as high quality as possible to maximize the final image quality.

From the highest level point of view, there is a single image processing chain. The image is taken from the Image Sensor, processed, and sent to the Printer Module for subsequent printout and/or storage.

We break the high level process into two image processing chains, each with a number of steps:

- Image Capture Chain
- Image Enhancement Chain

The Image Capture Chain is concerned with capturing the image from the Image Sensor and storing it locally within the Camera Module. The Image Enhancement Chain is concerned with taking the stored image and producing a high quality image for transmission to the Printer Module.

This section describes an implementation independent image processing chain that meets the quality requirements of PenPrint. At this stage, we are not considering *exactly how* the processing is performed in terms of hardware, but rather *what* must be done. These functions must be mapped onto various units within the PPCIP (see Section 24 on page 122).

Regardless of the PPCIP implementation, there are a number of constraints:

- The input image is a CFA based contone RGB image.
- The output image is an 850 × 534 contone L*a*b* image in a form acceptable for a PenPrint Printer Module.

23.1 IMAGE CAPTURE CHAIN

The Image Capture Chain is responsible for taking an image from the Image Sensor and storing it locally within the Camera Module. The Image Capture Chain is illustrated in Figure 72, with subsequent sections detailing the sub-components.

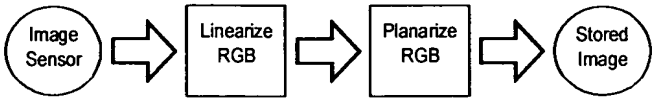


Figure 72. Image Capture Chain

23.1.1 Image Sensor

The input image comes from an image sensor. Although a variety of image sensors are available, we only consider the Bayer color filter array (CFA). The Bayer CFA has a number of attributes which are defined here.

The image captured by the CMOS sensor (via a taking lens) is assumed to have been sufficiently filtered so as to remove any aliasing artifacts. The sensor itself has an aspect ratio of approximately 3:2, with a resolution of 850 × 534 samples to match the image resolution of the final output image. The most likely pixel arrangement is the Bayer color filter array (CFA), with each 2×2 pixel block arranged in a 2G mosaic as shown in Figure 73:

Figure 73. Bayer CFA 2G Mosaic Pattern

Each contone sample of R, G, or B (corresponding to red, green, and blue respectively) is 10-bits. Note that each pixel of the mosaic contains information about only *one* of R, G, or B. Estimates of the missing color information must be made before the image can be printed out.

The CFA is considered to perform adequate fixed pattern noise (FPN) suppression.

23.1.2 Linearize RGB

The image sensor is unlikely to have a completely linear response. Therefore the 10-bit RGB samples from the CFA must be considered to be non-linear. These non-linear samples are translated into 8-bit linear samples by means of lookup tables (one table per color).

Pixels from the CFA lines 0, 2, 4 etc. index into the R and G tables, while pixels from the CFA lines 1, 3, 5 etc. index into the G and B tables. This is completely independent of the orientation of the camera. The process is shown in Figure 74. The total amount of memory

required for each lookup table is 2^{10} × 8-bits. The 3 lookup tables therefore require a total of ***3 KBytes*** (3×2^{10} bytes).

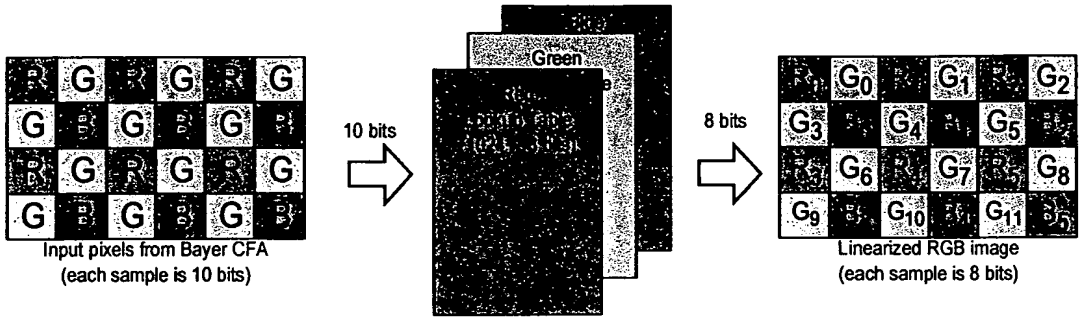


Figure 74. Linearize RGB

23.1.3 Planarize RGB

The pixels obtained from the CFA have their color planes interleaved due to the nature of the Bayer mosaic of pixels. By this we mean that on even horizontal lines, one red pixel is followed by a green pixel and then by another red pixel - the different color planes are interleaved with each other. In some image processing systems, an interleaved format is highly useful. However in the PenPrint Camera Module processing system, the algorithms are more efficient if working on planar RGB.

A planarized image is one that has been separated into its component colors. In the case of the CFA RGB image, there are 3 separate images: one image containing only the red pixels, one image containing only the blue pixels, and one image containing only the green pixels. Note that each plane only represents the pixels of that color which were *actually sampled.* No resampling is performed during the planarizing process. As a result, the R, G and B planes are not registered with each other, and the G plane is twice as large as either the R or B planes. The process is shown in Figure 75.

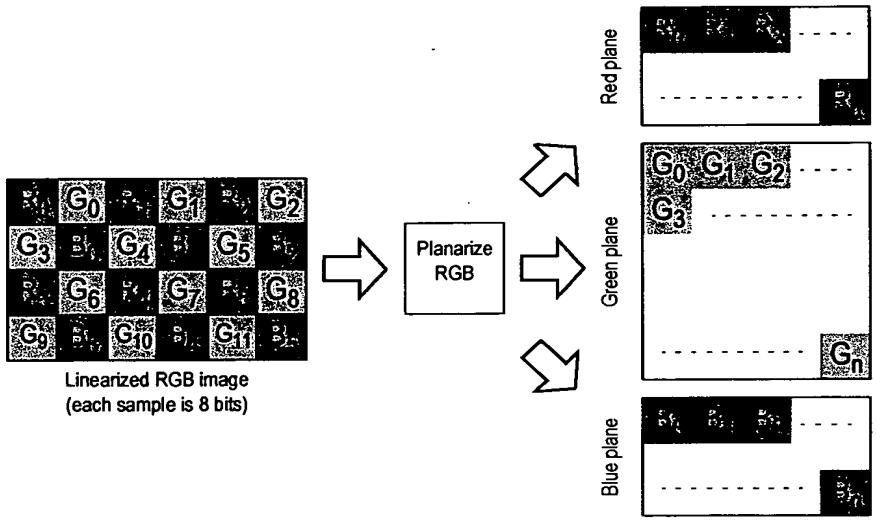


Figure 75. Planarize RGB

The actual process is quite simple - depending on the color of the pixels read in, the output pixels are sent to the next position in the appropriate color plane's image (therefore in the same orientation as the CFA).

The red and blue planar images are exactly one quarter of the size of the original CFA image. They are exactly half the resolution in each dimension. The red and blue images are therefore 425 x 267 pixels each, with the red image implicitly offset from the blue image by one pixel in CFA space (850 x 534) in both the x and y dimensions.

Although the green planar image is half of the size of the original CFA image, it is not set out as straightforwardly as the red or blue planes. The reason is due to the checkerboard layout of green. On one line the green is every *odd* pixel, and on the next line the green is every *even* pixel. Thus alternate lines of the green plane represent odd and even pixels within the CFA image. Thus the green planar image is 425 x 534 pixels. This has ramifications for the resampling process (see Section 23.2.4 on page 115 below).

23.1.4 Stored Image

Each color plane of the linearized RGB image is written to memory for temporary storage. The memory can be RAM since the image will be retained in the Printer Module or Memory Module after the power has been shut off.

The total amount of memory required for the entire planarized linear RGB image is 455,600 bytes (approximately 0.5 MB) arranged as follows:

- R: 425 x 267 = 113,475 bytes
- B: 425 x 267 = 113,475 bytes
- G: 425 x 537 = 226,950 bytes

23.2 IMAGE ENHANCEMENT CHAIN

The Image Enhancement Chain is concerned with taking the stored image and producing a high quality L*a*b* image for transmission to the Printer Module. The Image Enhancement Chain is necessary due to the specific attributes of the CFA image. There are a number of steps required in the image processing chain in order to produce high quality images from CFA captured images. The Image Enhancement Chain contains the functions:

- Gather Statistics
- White Balance
- Range Expansion
- Resample
- Convert to L*a*b*
- Sharpen

Figure 76 illustrates the Image Enhancement Chain.

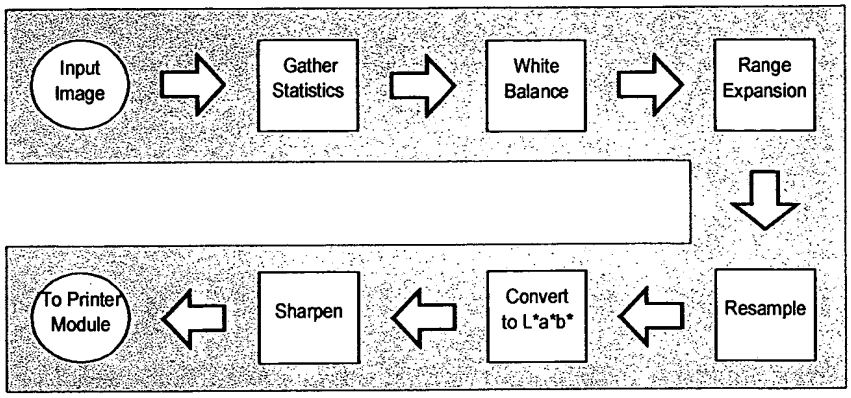


Figure 76. Image Enhancement Chain

23.2.1 Input Image

The input image is a linearized RGB image stored in planar form, as stored by the Image Capture Chain described in Section 23.1.4 on page 111.

23.2.2 Gather Statistics

A number of statistics regarding the *entire image* need to be gathered before processes like white balance and range expansion can be performed. These statistics can be gathered separately for the red, green, and blue planar images.

23.2.2.1 Build Histogram

The first step is to build a histogram for each 8-bit value of the color plane. Each 850 × 534 CFA image contains a total of:

- 113,475 red pixels (min 17-bit counter required)
- 113,475 blue pixels (min 17-bit counter required)
- 226,950 green pixels (min 18-bit counter required)

Therefore a single 256 × 18 bit table is required to hold the histogram.

The process of building the histogram is straightforward, as illustrated by the following pseudocode:

```
For I = 0 to 255
   Entry[I] = 0
EndFor
For Pixel = ImageStart to ImageEnd
   p = Image[Pixel]
   Entry[p] = Entry[p]+1
EndFor
```

23.2.2.2 Determine High and Low Thresholds

Once the histogram has been constructed for the color plane, it can be used to determine a high and low threshold. These thresholds can be used for automating later white balance and range expansion during the print process.

Basing the thresholds on the number of pixels from the histogram, we consider the *n% darkest pixels* to be expendable and therefore equal. In the same way, we consider the *n% lightest pixels* to be expendable and therefore equal. The exact value for *n* is expected to be about 5%, but will depend on the CFA response characteristics.

The process of determining the *n%* darkest values is straightforward. It involves stepping through the color plane's histogram from the count for 0 upwards (i.e. 0, 1, 2, 3 etc.) until the *n%* total is reached or we have travelled further than a set amount from 0. The highest of these values is considered the low threshold of the color plane. Although there is a difference between these darkest values, the difference can be considered expendable for the purposes of range expansion and color balancing.

The process of determining the *n%* lightest values is similar. It involves stepping through the color plane's histogram from the count for 255 downwards (i.e. 255, 254, 253 etc.) until the *n%* total is reached or until we have travelled further than a set amount from 255. The lowest of these values is considered the high threshold of the color plane. Although there is a difference between these lightest values, the difference can be considered expendable for the purposes of range expansion and color balancing.

The reason for stopping after a set distance from 0 or 255 is to compensate for two types of images:

- where the original dynamic range is low, or
- where there is no white or black in an image

In these two cases, we don't want to consider the entire *n%* of upper and lower values to be expendable since we have a low range to begin with. We can safely set the high and low thresholds to be outside the range of pixel values actually sampled. The exact distance will depend on the CFA, but will be two constants.

A sample color range for a color plane is shown in Figure 77. Note that although the entire 0-255 range is possible for an image color plane's pixels, this particular image has a smaller range. Note also that the same n% histogram range (represented by the blue bands) is represented by a larger range in the low end than in the high end. This is because the histogram must contain more pixels with high values closer together compared to the low end.

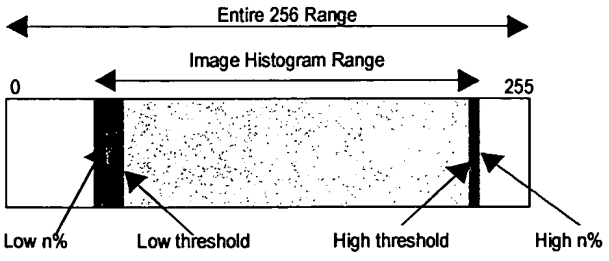


Figure 77. Sample Range for a Single Color Plane

The high and low thresholds must be determined for each color plane individually. This information will be used to calculate range scale and offset factors to be used in the later white balance and range expansion process.

The following pseudocode illustrates the process of determining either of the two thresholds (to find the low threshold, `StartPosition = 255`, and `Delta = 1`. To find the high threshold, `StartPosition = 0` and `Delta = -1`). The pseudocode assumes that `Threshold` is an 8-bit value that wraps during addition.

```
Threshold = StartPosition
Total = 0
TotalDelta = 0
While ((TotalDelta < MaxDelta) AND (Total < MaxPixels))
   Threshold = Threshold + Delta
   Total = Total + Entry[Threshold]
   TotalDelta = TotalDelta + 1
EndWhile
Return Threshold
```

23.2.3 White Balance and Range Expansion

A photograph is seldom taken in ideal lighting conditions. Even the very notion of "perfect lighting conditions" is fraught with subjectivity, both in terms of photographer and subject matter. However, in all cases, the subject matter of a photograph is illuminated by light either from a light source (such as the sun or indoor lighting), or its own light (such as a neon sign).

In most lighting conditions, what may appear to the photographer as "white" light, is usually far from white. Indoor lighting for example, typically has a yellow cast, and this yellow cast will appear on an uncorrected photograph. To most people, the yellow cast on the final uncorrected photograph is wrong. Although it may match the viewing conditions at the time the photograph was taken, it does not match the *perceived* color of the object. It is therefore crucial to perform white balance on a photograph before printing it out.

In the same way, an image can be perceived to be of higher quality when the dynamic range of the colors is expanded to match the full range in each color plane. This is particularly useful to do before an image is resampled to a higher resolution. If the dynamic range is higher, intermediate values can be used in interpolated pixel positions, avoiding a stepped or blocky image. Range expansion is designed to give the full 256 value range to those values actually sampled. In the best case, the lowest value is mapped to 0, and the highest value is mapped to 255. All the intermediate values are mapped to proportionally intermediate values between 0 and 255.

Mathematically, the operation performed is a translation of LowThreshold to 0 followed by a scale. The formula is shown here:

$$Pixel' = (Pixel - LowThreshold) \times RangeScaleFactor$$

$$\text{where} \quad RangeScaleFactor = \frac{256}{(HighThreshold - LowThreshold)}$$

RangeScaleFactor should be limited to a maximum value to reduce the risk of expanding the range too far. For details on calculating *LowThreshold*, see "Gather Statistics" on page

112. These values (*LowThreshold* and *RangeScaleFactor*) will be different for each color plane, and only need to be calculated once per image.

Both tasks can be undertaken simultaneously, as shown in Figure 78:

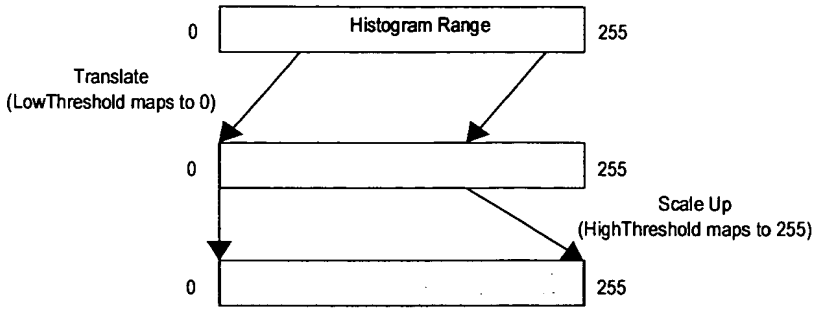


Figure 78. Performing White Balance and Range Expansion

Since this step involves a scaling process, we can be left with some fractional component in the mapped value e.g. the value *12* may map to *5.25*. Rather than discard the fractional component, we pass a 10 bit result (8 bits of integer, 2 of fraction) on to the next stage of the image processing chain. We cannot afford the memory to store the *entire* image at more than 8-bits, but we can make good use of the higher resolution in the resampling stage. Consequently the input image is 8-bits, and the output image has 10-bits per color component. The logical process is shown in Figure 79.

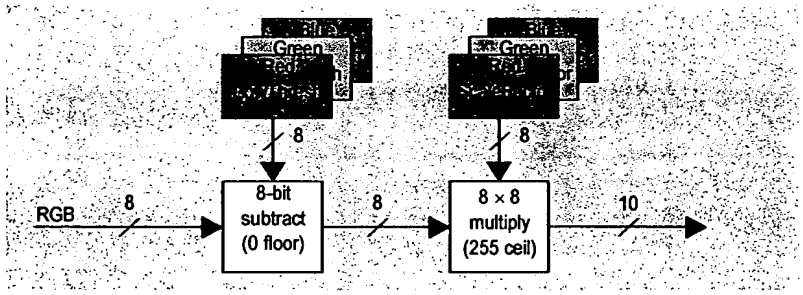


Figure 79. White Balance and Range Expansion

It is important to have a floor of 0 during the subtraction so that all values below *LowThreshold* to be mapped to 0. Likewise, the multiplication must have a ceiling of 255 for the integer portion of the result so that input values higher than *HighThreshold* will be mapped to 255.

23.2.4 Resample

The CFA only provides a single color component per pixel (x,y) coordinate. To produce the final image we need to have the other color component values at each pixel. With one color per pixel, we may have the red component for a particular position, but we need to estimate blue and green. Or we may have green, and need to estimate red and blue.

To decide how best to resample, it is best to consider each color plane in relation to the full CFA resolution (no rotation). This is shown in Figure 80.

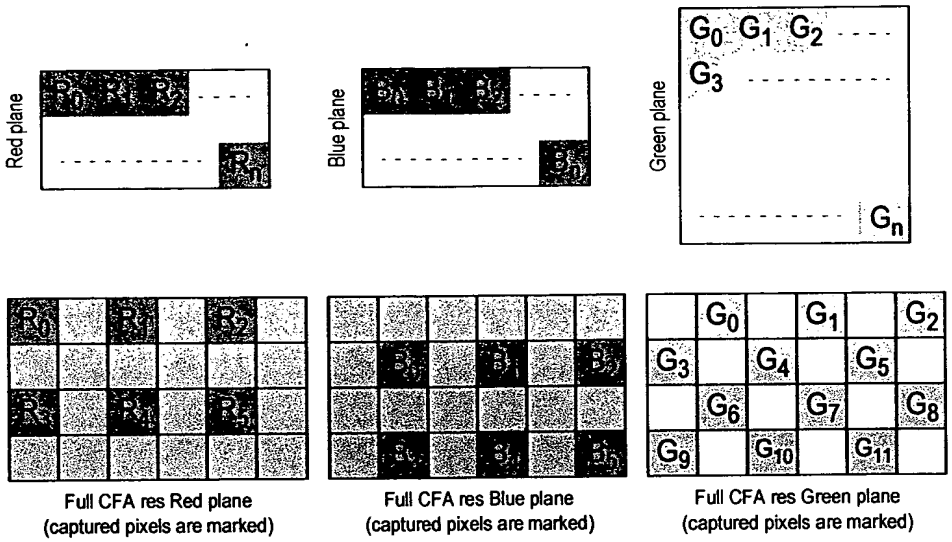


Figure 80. Color Plane Pixels in Relation to Full CFA Resolution

23.2.4.1 Red and Blue

Looking at the red and blue planes (see Figure 80), the full CFA resolution version of the color plane can be created by scaling up the number of sampled pixels in each dimension by 2. The intermediate pixels can be generated by means of a reconstruction filter (such as a Lanczos or Exponential filter). Only one dimension in the kernel is required, since the kernel is symmetric. Since red and blue have different offsets in terms of their initial representation within the CFA sample space, the initial positions in the kernel will be different.

The following relationship holds for red and blue mapping of output coordinates (in 534 space) to input coordinates:

$$\left.\begin{aligned} x' &= \left(\frac{x}{ops}\right) \\ y' &= \left(\frac{y}{ops}\right) \end{aligned}\right\} \quad \text{where:}$$

x, y = coordinate in output res space

x', y' = coordinate in input space

$ops = 2$ = output res pixels per input space sample

The number of output res pixels per sample, *ops*, is a constant value of 2. This means that given a starting position in input space, we can generate a new line of CFA resolution pixels by adding a Δx and Δy of 1/2 and 0 respectively 533 times. The fractional part of x and y in input space can be directly used for looking up the kernel coefficients for image reconstruction and resampling. Since Δx is 1/2, we only require 2 sets of kernel coefficients.

Since the red and blue planes are scaled *up*, there will not be any aliasing artifacts introduced by the resampling process.

23.2.4.2 Green

The green plane cannot be simply scaled up in the same way as red or blue, since each line of the green plane represents different pixels - either the odd or even pixels on alternate lines. Although in terms of the number of pixels it is representative to say the green image is 425 × 534, the image could equally be said to be 850 × 267. This confusion arises because of the checkerboard nature of the green pixels, where the distance between pixels is not equal in x and y dimensions, and does not map well to image reconstruction or resampling. The number of interpolation methods used by other systems for green plane reconstruction is testimony to this - from nearest neighbor replication to linear interpolation to bi-linear interpolation and heuristic reconstruction.

The mapping of output coordinates (in 534 space) to input coordinates is conceptually the same for green as it is for red and blue. For the green plane the following relationship holds:

$$\left.\begin{aligned} x' &= \left(\frac{x}{ops}\right) \\ y' &= \left(\frac{y}{ops}\right) \end{aligned}\right\} \quad \text{where:}$$

x, y = coordinate in output res space
x', y' = coordinate in input space
ops = 1 = output res pixels per input space sample

Setting the number of output res pixels per sample, *ops*, to 1 allows the direct usage of coordinates in CFA resolution input space. However, once we have a coordinate in CFA resolution input space, we cannot perform image reconstruction and resampling on the samples in the same way as red or blue due to the checkerboard nature of the green plane.

Instead, for the purposes of high quality image reconstruction and resampling, we can consider the green channel to be an image rotated by 45 degrees. When we look at the pixels in this light, as shown in Figure 81, a high quality image reconstruction and resampling method becomes clear.

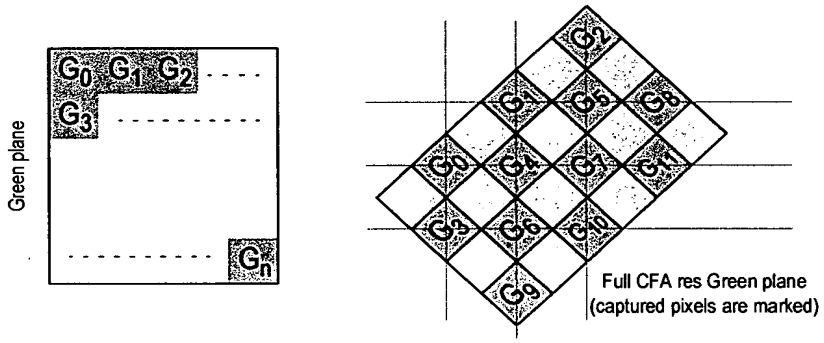


Figure 81. Considering the Green Plane as Rotated 45 Degrees

Looking at Figure 81, the distance between the sampled pixels in the X and Y directions is now equal. The actual distance between sampled pixels is $\sqrt{2}$, as illustrated in Figure 82.

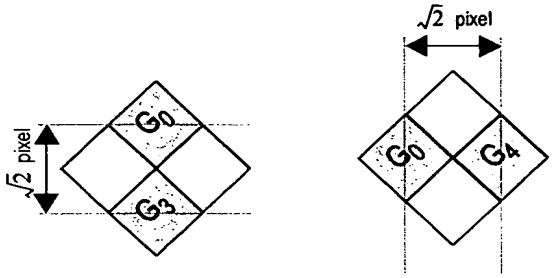


Figure 82. Calculating the Distance between Rotated Pixels

The solution for the green channel then, is to perform image reconstruction and resampling *in rotated space*. Although the same reconstruction filter is used as for resampling red and blue, the kernel should be different. This is because the relationship between the sampling rate for green and the highest frequency in the signal is different to the relationship for the red and blue planes. In addition, the kernel should be normalized so that the $\sqrt{2}$ distance between samples becomes 1 as far as kernel coordinates go (the unnormalized distances between resampling coordinates must still be used to determine whether aliasing will occur however). Therefore we require two transformations:

- The first is to map unrotated CFA space into rotated CFA space. This can be accomplished by multiplying each ordinate by $1/\sqrt{2}$, since we are rotating by 45 degrees ($\cos 45 = \sin 45 = 1/\sqrt{2}$).
- The second is to scale the coordinates to match the normalized kernel, which can be accomplished by multiplying each ordinate by $1/\sqrt{2}$.

These two transformations combine to create a multiplication factor of 1/2. Consequently, as we advance in unrotated CFA space x by *k*, we increase by k/2 in kernel x, and decrease by k/2 in kernel y. Similarly, as we advance in y by *k*, we increase by k/2 in kernel x and increase by k/2 in kernel y.

The relationships between these different coordinate systems can be illustrated by considering what occurs as we generate a line of output pixels from a CFA space input image. Given a starting y ordinate in CFA input space, we begin at x=0, and advance 850 times by 1, generating a new output pixel at each new location. The movement in unrotated CFA space by 1 can be decomposed into a movement in x and a movement in y in rotated CFA space. The process is shown in Figure 83.

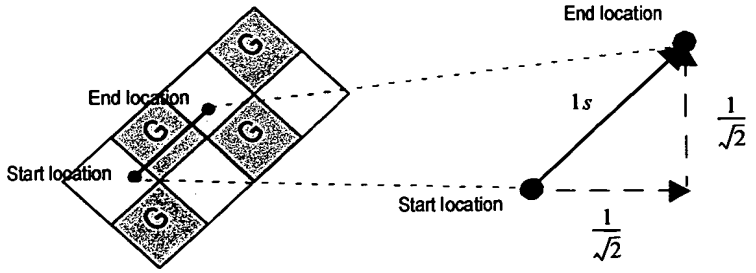


Figure 83. Mapping movement in unrotated CFA space to rotated CFA space

Since cos45 = sin45 = $1/\sqrt{2}$, movement in unrotated CFA space by 1 equates to equal movement in x and y by $1/\sqrt{2}$. This amount must now be scaled to match the normalized kernel. The scaling equates to another multiplication by $1/\sqrt{2}$. Consequently, a movement of 1 in unrotated CFA space equates to a movement of 1/2 in kernel x and kernel y.

Since the Δ of 1/2 is less than 1, we are scaling up, and therefore aliasing will not occur for green resampling. In addition, a Δ of 1/2 means that we only require 2 sets of kernel coefficients.

23.2.4.3 Reconstruction Filter for Red, Blue and Green

The exact reconstruction filter to be used will depend on a number of issues. There is always a trade off between the number of samples used in constructing the original signal, the time taken for signal reconstruction, and quality of the resampled image. A satisfactory trade-off in this case is 5 pixel samples from the dimension being reconstructed, centered around the estimated position X i.e. X-2, X-1, X, X+1, X+2. Due to the nature of reconstructing with 5 sample points, we only require 4 coefficients for the entry in the convolution kernel.

With generalized resampling, we create a kernel coefficient lookup table with *n* entries for each color component. Each entry has 4 coefficients. As we advance in output space, we map the changes in output space to changes in input space and kernel space. The most significant bits of the fractional component in the current kernel space are used to index into the kernel coefficients table. In this case, with red, green, and blue all requiring only 2 sets of kernel coefficients each, only 2 entries in the table is required: an entry for 0, and an entry for 1/2.

23.2.5 Convert to L*a*b*

The PenPrint Printer Module accepts images in the perceptually uniform CIE L*a*b* color space [1]. The Printer Module converts from L*a*b* to appropriate CMY values. Since a PenPrint Camera Module may be connected to different PenPrint Printer Modules, each potentially with its own ink characteristics, the most future-proof image format is L*a*b*. Converting to L*a*b* in the PPCIP also allows the image to be sharpened (see Section 23.2.6 on page 120).

To convert from RGB to L* (the luminance channel) we can average the minimum and maximum of R, G, and B as follows:

$$L^* = \frac{MIN(R, G, B) + MAX(R, G, B)}{2}$$

However, the conversion from RGB to a* and b* is more complex [1].

Rather than perform these transformations exhaustively, excellent results can be obtained via a tri-linear conversion based on 3 sets of 3D lookup tables. The lookup tables contain the resultant transformations for the specific entry as indexed by RGB. Three tables are required: one mapping RGB to L*, one mapping RGB to a*, and one mapping RGB to b*. Tri-linear interpolation can be used to give the final result for those entries not included in the tables. The process is shown in Figure 84.

Tri-linear interpolation requires reading 8 values from the lookup table, and performing 7 linear interpolations (4 in the first dimension, 2 in the second, and 1 in the third). High precision can be used for the intermediate values, although the output value is only 8 bits.

The size of the lookup table required depends on the linearity of the transformation. The recommended size for each table in this application is 17 × 17 × 17[1], with each entry 8 bits. A 17 × 17 × 17 table is 4913 bytes (less than 5KB).

To index into the 17-per-dimension tables, the 8-bit input color components are treated as fixed-point numbers (4:4). The 4 bits of integer give the index, and the 4 bits of fraction are used for interpolation.

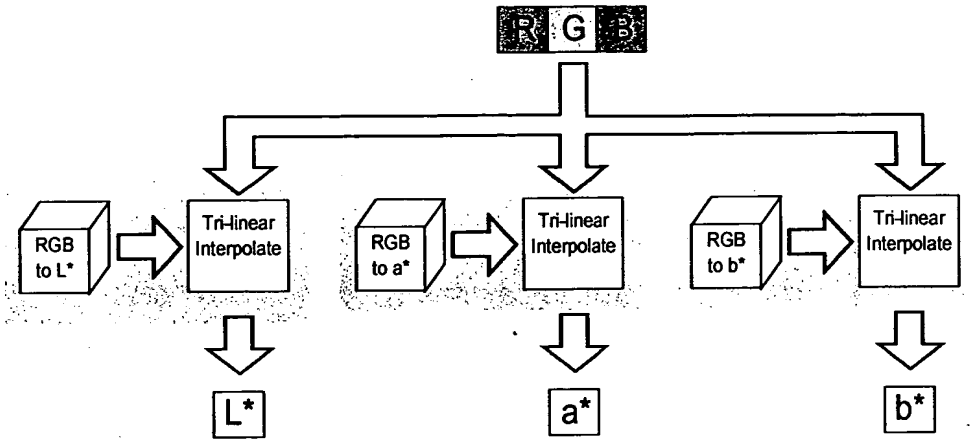


Figure 84. Conversion from RGB to L*a*b* by Trilinear Interpolation

23.2.6 Sharpen

The image captured by the CFA must be sharpened before being printed. Ideally, the sharpening filter should be applied in the CFA resolution domain, so it is performed at this stage in the image processing chain. Sharpening R, G, and B independently gives rise to color shifts. Sharpening should instead be applied to the luminance channel of an image, so that the hue and saturation of a given pixel will be unchanged. For this reason sharpening is performed *after* the conversion from RGB to L*a*b*.

Sharpening then, only affects the luminance channel by adding in a proportion of the high-pass-filtered version of the luminance. The process is shown in Figure 85.

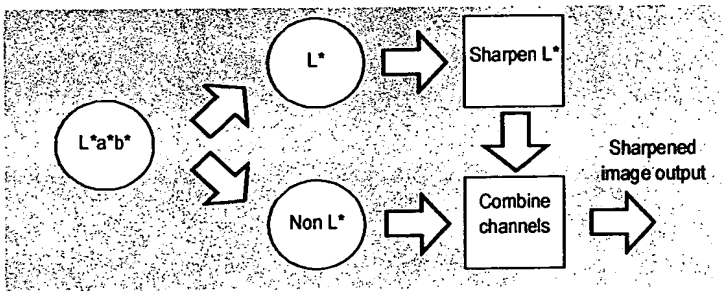


Figure 85. High level sharpening stages

1. Although a 17 × 17 × 17 table will give excellent results, it may be possible to get by with only a 9 × 9 × 9 conversion table (729 bytes). The exact size can be determined by simulation. The 5K conservative-but-definite-results approach was chosen for the purposes of this document.

To sharpen the image, a high pass filter is applied to the luminance information. Since we are filtering in CFA resolution space, a 3×3 convolution kernel will be sufficient to produce good results. The process of producing a single filtered L* pixel is shown in Figure 86.

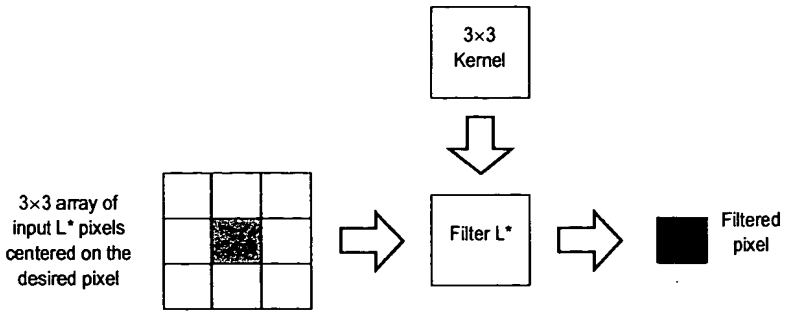


Figure 86. High-pass Filtering a Single Luminance Pixel with a 3×3 Kernel.

The actual kernel used can be any one of a set of standard highpass filter kernels. A basic but satisfactory highpass filter is shown in this implementation of the PPCIP in Figure 112.

Once the sharpening has been applied to the luminance pixel, the image can be recombined with the non-L* counterparts (a* and b*) before being transmitted to the PenPrint Printer Module.

24 PenPrint Camera Image Processor (PPCIP)

This chapter describes the PenPrint Camera Image Processor ASIC, a custom image processing chip contained within the PenPrint Camera Module.

24.1 High Level Internal Overview

The PPCIP is designed to be fabricated using a 0.25 micron CMOS process, with approximately 4 million transistors, almost half of which are flash memory or static RAM. This leads to an estimated area of 8mm^2. The estimated manufacturing cost is $2 in the year 2001. The PPCIP is a relatively straightforward design, and design effort can be reduced by the use of datapath compilation techniques, macrocells, and IP cores. The PPCIP contains:

- A low speed CPU/microcontroller core
- 0.5 MBytes of analog multi-level Flash memory (4-bits per cell)
- A CMOS Image Sensor Interface
- 8 KByte Flash memory for program storage
- 2 KByte RAM for program variable storage

The PPCIP is intended to run at a clock speed of approximately 24 MHz on 3V externally and 1.5V internally to minimize power consumption. The actual operating frequency will be an integer multiple of the PenPrint Serial Bus operating frequency. The CPU is intended to be a simple micro-controller style CPU, running at about 1 MHz. Both the CPU and CMOS Sensor Interface can be vendor supplied cores.

24.2 CPU Core and Memory

24.2.1 CPU Core

The PPCIP incorporates a simple micro-controller CPU core to synchronize the image capture and printing image processing chains and to perform general operating system duties. A wide variety of CPU cores are suitable: it can be any processor core with sufficient processing power to perform the required calculations and control functions fast enough to met consumer expectations.

Since all of the image processing is performed by dedicated hardware, the CPU does not have to process pixels. As a result, the CPU can be extremely simple. An example of a suitable core is a Philips 8051 micro-controller running at about 1 MHz.

There is no need to maintain instruction set continuity between different Camera Module models. Different PPCIP chip designs may be fabricated by different manufacturers, without requiring to license or port the CPU core. This device independence avoids the chip vendor lock-in such as has occurred in the PC market with Intel.

Associated with the CPU Core is a Program ROM and a small Program Scratch RAM.

The CPU communicates with the other units within the PPCIP via memory-mapped I/O. Particular address ranges map to particular units, and within each range, to particular registers within that particular unit. This includes the serial and parallel interfaces.

24.2.2 Program ROM

A small Program Flash ROM is incorporated into the PPCIP. The ROM size depends on the CPU chosen, but should not be more than about 8KB.

24.2.3 Program RAM

Likewise, a small scratch RAM area is incorporated into the PPCIP. Since the program code does not have to manipulate images, there is no need for a large scratch area. The RAM size depends on the CPU chosen (e.g. stack mechanisms, subroutine calling conventions, register sizes etc.), but should not be more than about 2KB.

24.2.4 CPU Memory Decoder

The CPU Memory Decoder is a simple decoder for satisfying CPU data accesses. The Decoder translates data addresses into internal PPCIP register accesses over the internal low speed bus, and therefore allows for memory mapped I/O of PPCIP registers.

24.3 COMMUNICATION INTERFACES

24.3.1 PenPrint Serial Bus Interface

This is a standard PenPrint Serial Bus Interface connected to a PenPrint Serial Bus. Although the maximum speed on the bus is 12 MBits/sec, the maximum effective data transfer rate is 8MBits/sec due to protocol overhead and transmission redundancy.

The PenPrint Serial Bus is used to transmit commands and images between the various modules of the PenPrint system.

Since the PPCIP is responsible for transmitting an image to the PenPrint Printer Module, the transmission timing considerations are required for image processing parameters. The time taken to transmit a complete image (850 × 534 L*a*b*) is **1.36 seconds** (850 × 534 × 3 × 8 / 8,000,000).

24.3.2 Parallel Interface

The parallel interface connects the PPCIP to individual static electrical signals. The CPU is able to control each of these connections as memory-mapped I/O via the low-speed bus. (See Section 24.2.4 on page 123 for more details on memory-mapped I/O).

Table 28 shows the connections to the parallel interface.

Table 28. Connections to Parallel Interface

Connection	Direction	Pins
"Take" button	In	1
Timer on/off switch	In	1
	TOTAL	2

24.3.3 JTAG Interface

A standard JTAG (Joint Test Action Group) Interface is included in the PPCIP for testing purposes. Due to the complexity of the chip, a variety of testing techniques are required,

including BIST (Built In Self Test) and functional block isolation. An overhead of 10% in chip area is assumed for overall chip testing circuitry.

24.4 IMAGE RAM

The Image RAM is used to store the captured image. The Image RAM is analog multi-level Flash (4-bits per cell) so that the image is retained after the power has been shut off. Although Flash memory is not required, multilevel Flash memory requires fewer gates than RAM, and the use of 16-level Flash (4-bit per cell) is possible as the occasional bit error in the image is not fatal (compared to program code).

The total amount of memory required for the planarized linear RGB image is 500,000 bytes (approximately 0.5 MB) arranged as follows:

- R: 425 × 267 = 113,475 bytes
- B: 425 × 267 = 113,475 bytes
- G: 425 × 537 = 226,950 bytes

The image is written by the Image Capture Unit, and read by both the Image Histogram Unit and the Image Enhancement Unit. The CPU does not have direct random access to this image memory.

24.5 IMAGE CAPTURE UNIT

The Image Capture Unit contains all the functionality required by the Image Capture Chain, as described in Section 23.1 on page 109. The Image Capture Unit accepts pixel data via the Image Sensor Interface, linearizes the RGB data via a lookup table, and finally writes the linearized RGB image out to RAM in planar format. The process is shown in Figure 87.

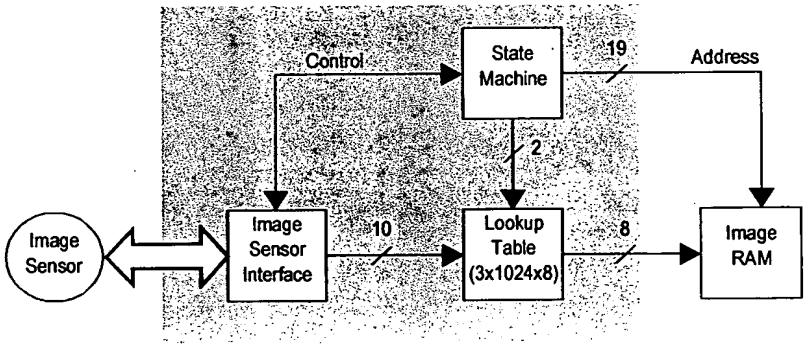


Figure 87. Image Capture Unit

24.5.1 Image Sensor Interface

The Image Sensor Interface (ISI) is a state machine that sends control information to the external CMOS Image Sensor, including frame sync pulses and pixel clock pulses in order to read the image. Most of the ISI is likely to be a sourced cell from the image sensor manufacturer. The ISI is itself controlled by the Image Capture Unit State Machine.

Although a variety of image sensors are available, we only consider the Bayer color filter array (CFA). The Bayer CFA has a number of attributes which are defined here.

The image captured by the CMOS sensor (via a taking lens) is assumed to have been sufficiently filtered so as to remove any aliasing artifacts. The sensor itself has an aspect ratio of approximately 3:2, with a resolution of 850 × 534 samples. The most likely pixel arrangement is the Bayer color filter array (CFA), with each 2 × 2 pixel block arranged in a 2G mosaic as shown in Figure 88:

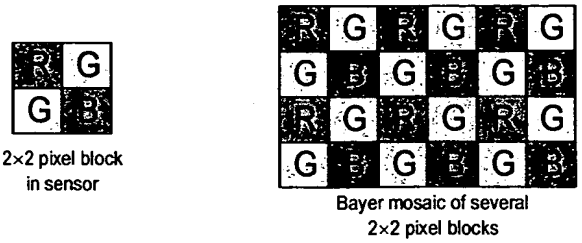


Figure 88. Bayer CFA 2G Mosaic Pattern

Each contone sample of R, G, or B (corresponding to red, green, and blue respectively) is 10-bits. Note that each pixel of the mosaic contains information about only *one* of R, G, or B. Estimates of the missing color information must be made before the image can be printed out.

The CFA is considered to perform some amount of fixed pattern noise (FPN) suppression. Additional FPN suppression may required.

24.5.2 Lookup Table

The lookup table is a ROM mapping the sensor's RGB to a linear RGB. It matches the Linearize RGB process described in Section 23.1.2 on page 109. As such, the ROM is 3 KBytes (3 × 1024 × 8-bits). 10 bits of address come from the ISI, while the 2 bits of Table-Select are generated by the Image Capture Unit's State Machine.

24.5.3 State Machine

The Image Capture Unit's State Machine generates control signals for the Image Sensor Interface, and generates addresses for linearizing the RGB and for planarizing the image data.

The control signals sent to the ISI inform the ISI to start capturing pixels, stop capturing pixels etc.

The 2-bit address sent to the Lookup Table matches the current line being read from the ISI. For even lines (0, 2, 4 etc.), the 2-bit address is Red, Green, Red, Green etc. For odd lines (1, 3, 5 etc.), the 2-bit address is Green, Blue, Green, Blue. This is true regardless of the orientation of the camera.

The 21-bit address sent to the Image RAM is the write address for the image. Three registers hold the current address for each of the red, green, and blue planes. The addresses increment as pixels are written to each plane.

24.5.4 Registers

The Image Capture Unit contains a number of registers:

Table 29. Registers in Image Capture Unit

Name	Bits	Description
MaxPixels	10	Number of pixels each row
MaxRows	10	Number of rows of pixels in image
CurrentPixel	10	Pixel currently being fetched
CurrentRow	10	Row currently being processed
NextR	19	The address in Image RAM to store the next Red pixel. Set to start address of red plane before image capture. After image capture, this register will point to the byte *after* the red plane.
NextG	19	The address in Image RAM to store the next Green pixel. Set to start address of green plane before image capture. After image capture, this register will point to the byte *after* the green plane.
NextB	19	The address in Image RAM to store the next Blue pixel. Set to start address of blue plane before image capture. After image capture, this register will point to the byte *after* the blue plane.
EvenEven	2	Address to use for even rows / even pixels
EvenOdd	2	Address to use for even rows / odd pixels
OddEven	2	Address to use for odd rows / even pixels
OddOdd	2	Address to use for odd rows / odd pixels
Go	1	Writing a 1 here starts the capture. Writing a 0 here stops the image capture. A 0 is written here automatically by the state machine after MaxRows of MaxPixels have ben captured.

In addition, the Image Sensor Interface contains a number of registers. The exact registers will depend on the Image Sensor chosen.

24.6 IMAGE HISTOGRAM UNIT

The Image Histogram Unit (IHU) is designed to generate histograms of images as required by the Print Image Processing Chain described in Section 23.2.2 on page 112. The IHU only generates histograms for planar format images with samples of 8 bits each.

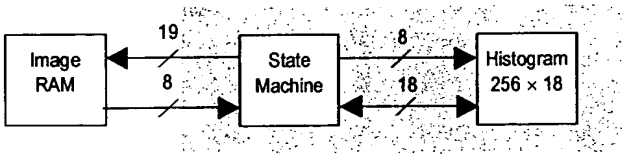


Figure 89. Image Histogram Unit

The Image Histogram Unit is typically used three times per print. Three different histograms are gathered, one per color plane. Each time a histogram is gathered, the results are

analyzed in order to determine the low and high thresholds, scaling factors etc. for use in the remainder of the print process. For more information on how the histogram should be used, see Section 23.2.2.2 on page 113 and Section 23.2.3 on page 114.

24.6.1 Histogram RAM

The histogram itself is stored in a 256-entry RAM, each entry being 18 bits. The histogram RAM is only accessed from within the IHU. Individual entries are read from and written to as 18-bit quantities.

24.6.2 State Machine and Registers

The State Machine follows the pseudocode described in Section 23.2.2.1 on page 112. It is controlled by the registers shown in Table 30.

Table 30. Registers in Image Histogram Unit

Name	Bits	Description
TotalPixels	19	The number of pixels to count (decrements until 0)
StartAddress	19	Where to start counting from
PixelsRemaining	19	How many pixels remain to be counted
PixelValue	8	A write to this register loads PixelCount with the PixelValue entry from the histogram.
PixelCount	18	The number of PixelValue pixels counted in the current histogram. It is valid after a write to PixelValue.
ClearCount	1	Determines whether the histogram count will be cleared at the start of the histogram process. A 1 causes the counts to be cleared, and a 0 causes the counts to remain untouched (i.e. the next histogram adds to the existing counts).
Go	1	Writing a 1 here starts the histogram process. Writing a 0 here stops the histogram process. A 0 is written here automatically by the state machine after TotalPixels has counted down to 0.

The typical usage of the registers is to set up TotalPixels with the total number of pixels to include in the count (e.g. 113,475 for red), StartAddress with the address of the red plane, ClearCount with 1, and write a 1 to the Go register. Once the count has finished, the individual values in the histogram can be determined by writing 0-255 to PixelValue and reading the corresponding PixelCount.

24.7 IMAGE ENHANCEMENT UNIT

The Image Enhancement Unit (IEU) is an implementation of most of the Image Enhancement Chain described in Section 23.2 on page 111.

From the simplest point of view, the IEU provides the interface between the Image RAM and the PenPrint Serial Bus Interface, as shown in Figure 52. The IEU takes a planarized linear RGB obtained from a CFA format captured image from the ImageRAM, and pro-

duces a fully populated L*a*b* image of resolution 850 × 534 for subsequent transmission to the PenPrint Printer Module via the PenPrint Serial Interface.

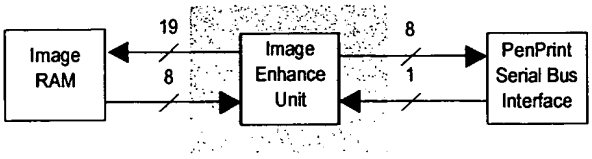


Figure 90. Interface of Print Generator Unit

The Image Enhancement Chain described in Section 23.2 on page 111 contains the functions:

- Gather Statistics
- Rotate Image
- White Balance
- Range Expansion
- Resample
- Sharpen
- Convert to L*a*b*

The IEU contains all of these functions with the exception of Gather Statistics. To perform the Gather Statistics step, the CPU calls the Image Histogram Unit three times (once per color channel), and applies some simple algorithms (see Section 23.2.2.2 on page 113). The remainder of the functions are the domain of the IEU for reasons of accuracy and speed: accuracy, because there would be too much memory required to hold the entire image at high color accuracy, and speed, because a simple microcontroller CPU running at low speed cannot produce the fully populated L*a*b* image in a reasonable time.

In terms of speed, the IEU is producing data for the PenPrint Serial Bus Interface. Given that the Penprint Serial Bus Interface transmits at an effective rate of 8 MHz[1], and the PPCIP runs at 24 MHz, it takes the serial interface 72 cycles to transmit the 24 bits for each pixel (3 colors @ 8 bits per color @ 3 cycles to transfer each bit). **The IEU therefore has 72 cycles to produce each L*a*b* triplet.**

The IEU takes as input a variety of parameters, including RGB to L*a*b* conversion tables, and constants for performing white balance and range expansion.

The IEU consists of 4 processes, all running in parallel. The first process performs White Balance and Range Expansion. The second process performs Resampling. The third performs color conversion, and the fourth performs sharpening. The output from the IEU is directly sent to the PenPrint Serial Bus Interface for transmission to the PenPrint Printer

1. Although the maximum speed on USB is 12 MBits/sec, the maximum effective data transfer rate is 8MBits/sec due to protocol overhead and transmission redundancy. The time taken to transmit a complete image (850 × 534 L*a*b*) is therefore 1.36 seconds (850 × 534 × 3 × 8 / 8,000,000).

Module. The processes are connected via buffers, each typically only a few bytes. An overview is shown in Figure 91.

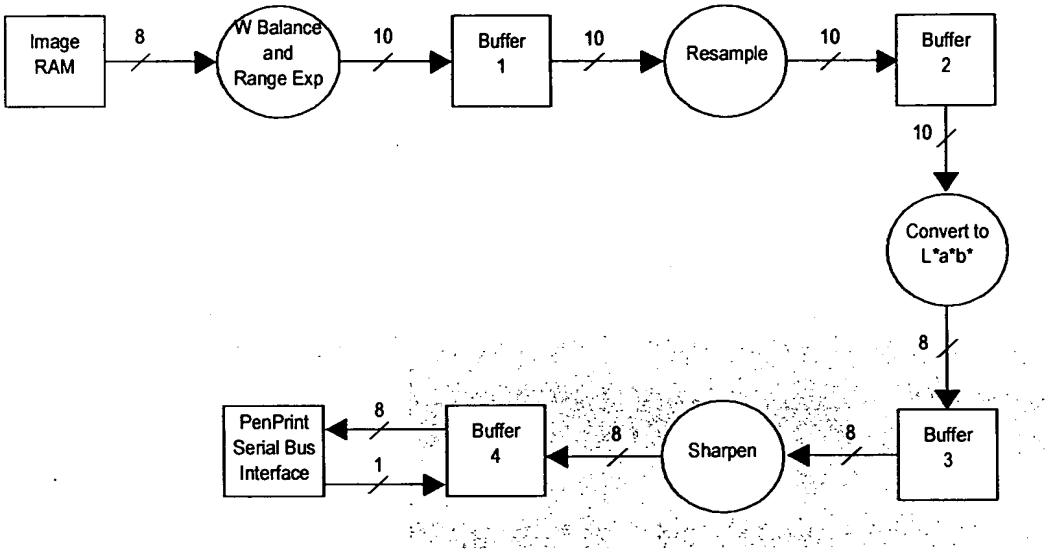


Figure 91. Image Enhancement Unit (shaded area)

The buffer sizes are summarized in Table 31.

Table 31. Buffer sizes for Image Enhancement Unit

Buffer	Size (bytes)	Composition of Buffer
Buffer 1	172.5	Red Buffer = 6 lines of 6 entries @ 10-bits each = 45 bytes Blue Buffer = 6 lines of 6 entries @ 10-bits each = 45 bytes Green Buffer = 11 lines of 6 entries @ 10-bits each = 82.5 bytes
Buffer 2	22.5	2 sets of 3 RGB pixels = 2 × 3 × 3 entries @ 10-bits each = 180 bits
Buffer 3	20	5 × 4 RAM 3 lines of 4 entries of L @ 8-bits each = 12 bytes 2 colors × 4 entries @ 8-bits each = 8 bytes
Buffer 4	6	2 L*a*b* pixels = 2 × 3 entries @ 8-bits each = 6 bytes
TOTAL	221	

Apart from a number of registers, some of the processes have significant lookup tables or memory components. These are summarized in Table 32.

Table 32. Memory requirements within IEU Processes

Unit	Size (bytes)	Composition of Requirements
White Balance / Range Expand	0	
Resample	36	3 kernels, each 2×4×12-bits
Convert to L*a*b*	14,739	3 conversion tables, each 17×17×17×8-bits

Table 32. Memory requirements within IEU Processes

Unit	Size (bytes)	Composition of Requirements
Sharpen	0	
TOTAL	14,775	

24.7.1 Buffer 1

Buffer 1 holds the white-balanced and range-expanded pixels at the original capture spatial resolution. Each pixel is stored with 10 bits of color resolution, compared to the image RAM image storage color resolution of 8 bits per pixel.

Buffer 1 is arranged as 3 separately addressable buffers - one for each color plane of red, green, and blue. A simple overview of the buffers is shown in Figure 92.

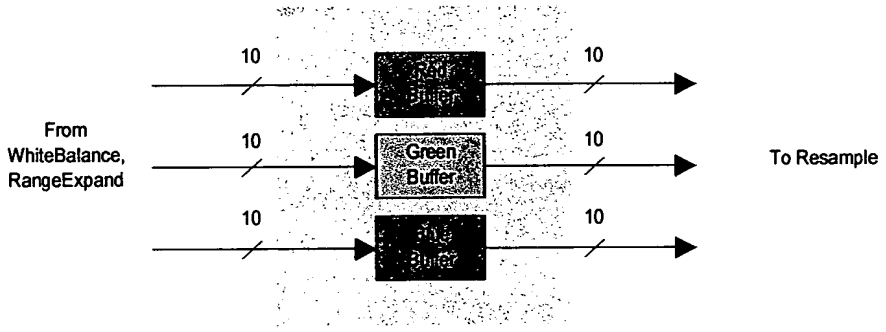


Figure 92. Structure of Buffer 1

During the course of 72 cycles, 16 entries are read from each of the 3 buffers 3 times by the Resampling process, and up to 29 new values are written to the 3 buffers (the exact number depends on the scale factor and the current sub-pixel position during resampling).

The buffers must be wide enough so that the reading and writing can occur without interfering with one another. During the read process, 4 pixels are read from each of 6 rows. On average each input pixel is used twice.

The green plane has a Δ value of 0.5 for resampling, indicating that 4 sample positions can be contained within 2 CFA pixel positions. However, each row of green samples only holds every alternate pixel. This means that only 4 samples are required per row (worst case is 4, not 3, due to a worst case initial position). Movement in Y indicates the requirement of an additional sample column, making 5. Finally, an additional sample column is required for writing. This gives a total of 6 samples per row. 7 rows are required for a single sample. To generate the 3 sets of RGB pixels for each X position, the maximum movement in Y will be 2 rows. Movement in X adds one sample row above and below. Consequently a total of 11 rows are required. For more details see Section 24.7.6.5 on page 143.

The red and blue planes have a Δ value of 0.5 for resampling, indicating that 4 locations can be contained within 2 samples. 4 samples per row are required for the resampling process, which is further increased to 6 samples to match the green plane (for startup pur-

poses). 6 rows are required to cater for movement in Y. For more details see Section 24.7.6.6 on page 148.

Each sub-buffer is implemented as a RAM with decoding to read or write a single 10-bit sample per cycle. The sub-buffers are summarized in Table 33, and consume less than 175 bytes.

Table 33. Sub-Buffer Summary

Buffer	Composition	Bits
Red Buffer	6 rows × 6 samples × 10-bits	360
Blue Buffer	6 rows × 6 samples × 10-bits	360
Green Buffer	11 rows × 6 samples × 10 bits	660
	TOTAL	1380

24.7.2 Buffer 2

Buffer 2 holds 3 complete sets of RGB pixel values (10-bits per color component) for a given CFA coordinate in a double buffered format (one set of 3 RGB pixels is being read by the Color Convert process, while the other is being written to 10-bits at a time by the Resample process). A simple overview of the buffers is shown in Figure 93.

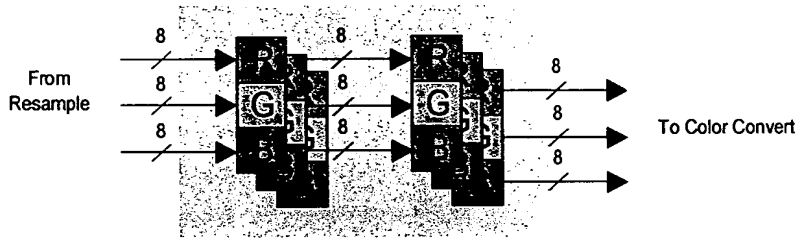


Figure 93. Structure of Buffer 2

The values are moved from the first RGB buffer to the second once the Color Convert and Resample processes have both finished. This can simply be achieved by a Select bit that is toggled, rather than physically transferring the data from one set of 9 bytes to the other.

24.7.3 Buffer 3

Buffer 3 accepts the output from the Color Convert process, where a complete L*a*b* pixel is generated from the RGB equivalent for a given pixel coordinate. Buffer 3 is used by the Sharpen process, which requires a 3 × 3 set of luminance values centered on the pixel being sharpened.

Consequently, during the sharpening process, there is need for access to the 3 × 3 array of luminance values, as well as the corresponding a*b* value for the center luminance pixel.

At the same time, the next 3 L*a*b* values must be calculated by from the RGB values by the Color Convert process. The logical view of accesses to Buffer 3 is shown in Figure 94.

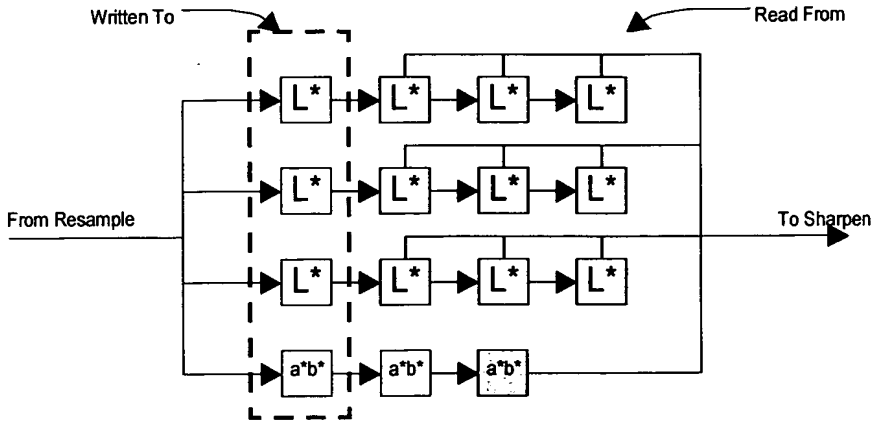


Figure 94. Logical Structure of Buffer 3

The actual implementation of Buffer 3 is simply as a 4 × 5 (20 entry) 8-bit RAM, with the addressing on read and write providing the effective shifting of values. A 2-bit column counter can be incremented with wrapping to provide a cyclical buffer, which effectively implements the equivalent of shifting the entire buffer's data by 1 column position. The fact that we don't require the fourth column of a*b* data is not relevant, and merely uses 2 bytes at the saving of not having to implement complicated shift and read/write logic. In a given cycle, the RAM can either be written to or read from. The read and write processes have 72 cycles in which to complete in order to keep up with the transmission process.

24.7.4 Buffer 4

Buffer 4 holds a complete L*a*b* pixel value (8-bits per color component) ready to be transmitted by the PenPrint Serial Bus Interface to the PenPrint Printer Module. Buffer 4 is a double buffered format (one L*a*b* pixel is being read by the PenPrint Serial Bus Interface process, while the other is being written to by the Sharpen process during a 72 cycle period). A simple overview of the buffers is shown in Figure 95.

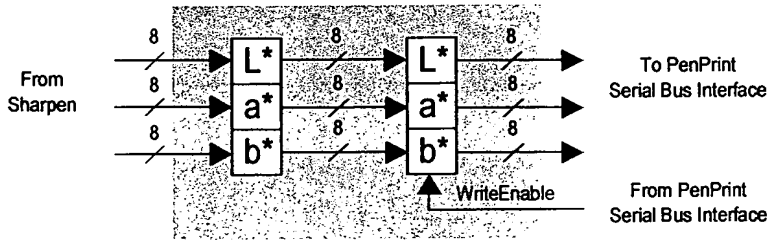


Figure 95. Structure of Buffer 4

The values are moved from the first buffer to the second once the PenPrint Serial Bus Interface has transmitted all 24 bits of L*a*b*. As with Buffer 2, the double buffering can simply be achieved by a Select bit that is toggled, rather than physically transferring the data from one set of 3 bytes to the other.The PenPrint Serial Bus Interface supplies the WriteEnable signal for the transfer.

24.7.5 White Balance and Range Expansion

The actual task of loading Buffer 1 from the Image RAM involves the steps of white balance and range expansion, as described by Section 23.2.3 on page 114. The pixels must be produced for Buffer 1 fast enough for their use by the Resampling process. This means that during a single group of 72 cycles, this unit must be able to read, process, and store 6 red pixels, 6 blue pixels, and 11 green pixels.

Once a given pixel has been read from the appropriate plane in the image store, it must be white balanced and its value adjusted according to the range expansion calculation defined in Section 23.2.3 on page 114. The process simply involves a single subtraction (floor 0), and a multiply (255 ceiling), both against color specific constants. The structure of this unit is shown in Figure 96.

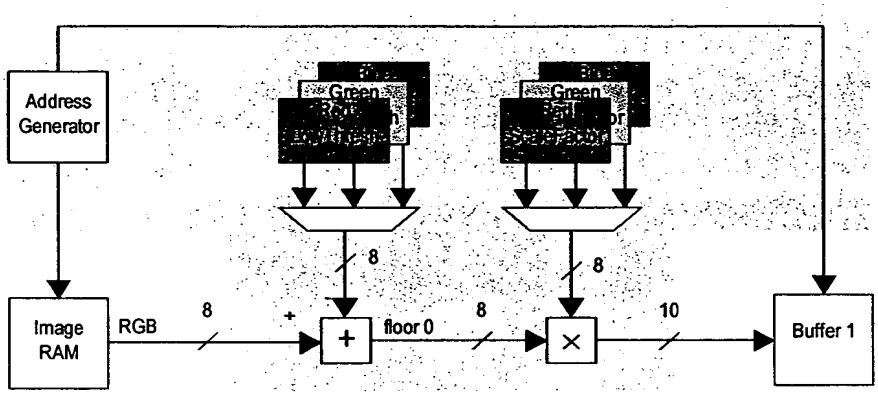


Figure 96. Rotate, White Balance and Range Expansion

The red, green and blue low thresholds, together with the red, green, and blue scale factors are determined by the CPU after generating the histograms for each color plane via the Image Histogram Unit (see Section 24.6 on page 126).

Depending on whether the current pixel being processed in the pipeline is red, green, or blue, the appropriate low threshold and scale factor is multiplexed into the subtract unit and multiply unit, with the output written to the appropriate color plane in Buffer 1.

The Subtract unit subtracts the 8-bit low Threshold value from the 8-bit Image RAM pixel value, and has a floor of 0. The 8-bit result is passed on to the specialized 8 × 8 multiply unit, which multiplies the 8-bit value by the 8-bit scale factor (8 bits of fraction, integer=1). Only the top 10 bits of the result are kept, and represent 8 bits of integer and 2 bits of fraction. The multiplier has a result ceiling of 255, so if any bit higher than bit 7 would have been set as a result of the multiply, the entire 8-bit integer result is set to 1s, and the fractional part set to 0.

Apart from the subtraction and multiply units, the majority of work in this unit is performed by the Address Generator, which is effectively the state machine for the unit. The address generation is governed by two factors: on a given cycle, only one access can be made to the Image RAM, and on a given cycle, only one access can be made to Buffer 1. Of the 72 available cycles, 3 sets of 16 cycles are used by the Resampler for reading Buffer 1. The actual usage is 3 sets of 24 cycles, with 16 reads followed by 8 wait cycles. That gives a total of 24 available cycles for 23 writes (6 red, 6 blue, 11 green). This means

the two constraints are satisfied if the timing of the writes to Buffer 1 coincide with the wait cycles of the Resampler.

24.7.5.1 Address Generation for Buffer 1

Once the resampling process is running, we are only concerned with writing to Buffer 1 during the period when the Resampler is not reading from it. Since the Resampler has 3 sets of 16 reads each 72 cycle period, there are a total of 24 cycles available for writing (cycles 16-23, 40-47, and 64-71). When the resampler is not running, we want to load up Buffer 1 as fast as possible, which means a write to Buffer 1 each cycle. Address Generation for Buffer 1 consequently runs off a state machine that takes these two cases into account. Whenever a value is loaded from ImageRAM, the adjusted value is written to the appropriate color in Buffer 1 one cycle later.

Address Generation for Buffer 1 therefore involves a single address counter for each of the red, blue and green sub-buffers. The initial address for `RedAdr`, `BlueAdr` and `GreenAdr` is 0 at the start of each line in each case, and after each write to Buffer 1, the address increments by 1, with wrapping at 36 or 66, depending on whether the buffer being written to is red, green or blue. Not all colors are written each 72-cycle period. A column of green will require replenishing at a faster rate than red or blue, for example.

The logic is shown in the following pseudocode:

```
If the color to write is Red
   Write to Red Buffer1 at RedAdr
   RedAdr = RedAdr + 1 mod 36
Else
If the color to write is Blue
   Write to Blue Buffer1 at BlueAdr
   BlueAdr = BlueAdr + 1 mod 36
Else
If the color to write is Green
   Write to Green Buffer1 at GreenAdr
   GreenAdr = GreenAdr + 1 mod 66
EndIf
```

24.7.5.2 Address Generation for Image RAM

Each color plane is read in a row-wise fashion. In addition, we allow edge pixel replication or constant color for reads outside image bounds.

At the start of each print line we must read the ImageRAM to load up Buffer 1 as fast as possible. This equates to a single access to a sample each cycle. Resampling can only occur once 5 columns have been loaded, which means 5 columns of 6, 6, and 11 samples, for a total of 115 cycles. Plus an extra cycle for the final value to be written out to Buffer 1 after being loaded from ImageRAM. To make the counting easier, we round up to 128 cycles.

After the first 128 cycles, the checking for the requirement to load the next column of samples for each of the 3 colors occurs each 72 cycles, with the appropriate samples loaded during the subsequent 72 cycles. However, the initial setting of whether to load during the first set of 72 cycles is always 1 for each color. This enables the final 6th column of each color within Buffer 1 to be filled.

At the end of each 72 cycle period, the `KernelXCarryOut` flag from each color plane of the Kernel Address Generator in the Resampler is checked to determine if the next column of samples should be read. Similarly, at the end of the line, the process restarts on the following line if the `KernelYCarryOut` flag is set.

Since each 'read' effectively becomes 6 or 11 reads to fill a column in Buffer 1, we keep a starting position in order to advance to the next 'read'. We also keep a coordinate value to allow the generation of out-of-bounds coordinates to enable edge pixel replication and constant color.

We consider the active image as being within a particular bounds, with certain actions to be taken when coordinates are outside the active area. The coordinates can either be before the image, inside the image, or after the image, both in terms of lines and pixels. This is shown in Figure 97, although the space outside the active area has been exaggerated for clarity:

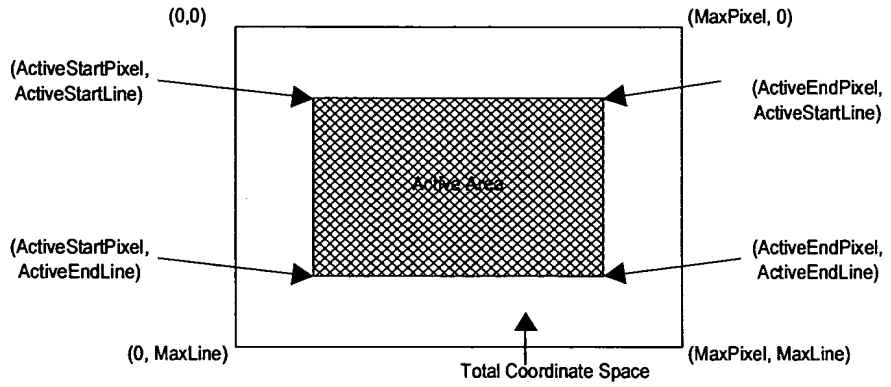


Figure 97. Active Image Area Within Generated Coordinate Space

Note that since we use (0, 0) as the start of coordinate generation, MaxPixel and MaxLine are also pixel and line counts. However, since address generation is run from kernel carry outs and AdvanceLine pulses these outer bounds are not required. Address generation for a line simply continues until the AdvanceLine pulse is received (once 534 sets of L*a*b* pixels have been transmitted along the PenPrint Serial Bus), and may involve edge replication, or constant colors for out of bounds.

If we have an address, `Adr`, of the current sample, and want to move to the next sample, either on the next line or on the same line, the sample's coordinate will change as expected, but the way in which the address changes depends on whether we are wrapping around the active image, and must produce edge pixel replication when needed.

We perform the actions in Table 34 as we advance in line or pixel. Looking at Table 34, the only time that `ADR` changes is by ΔPixel when PixelSense is 0, and by ΔLine when LineSense is 0. By following these simple rules `Adr` will be valid for edge pixel replica-

tion. Of course, if a constant color is desired for out of bounds coordinates, that value can be selected in instead of the value stored at the appropriate address.

Table 34. Actions to Perform when Advancing in Pixel or Line

Line[a]	Pixel[b]	Pixel Change	Line Change
-	-		
-	0	Adr = Adr + ΔPixel	
-	+		
0	-		Adr = Adr +ΔLine
0	0	Adr = Adr + ΔPixel	Adr = Adr +ΔLine
0	+		Adr = Adr +ΔLine
+	-		
+	0	Adr = Adr + ΔPixel	
+	+		

a. We compare the current Line ordinate with ActiveStartLine and ActiveEndLine.
If Line < ActiveStartLine, we call the value "-".
If ActiveStartLine ≤ Line < ActiveEndLine, we call the value "0".
If ActiveEndLine ≤ Line, we call the value "+".

b. We compare the current Pixel ordinate with ActiveStartPixel and ActiveEndPixel.
If Pixel < ActiveStartPixel, we call the value "-".
If ActiveStartPixel ≤ Line < ActiveEndPixel, we call the value "0".
If ActiveEndPixel ≤ Pixel, we call the value "+".

The logic for loading the set number of samples (either 6 or 11, depending on color) is shown in the following pseudocode:

```
line = FirstSampleLine
pixel = FirstSamplePixel
adr = FirstSampleAdr
Do N times (6 or 11)
   oldPixelSense = PixelSense(pixel)
   oldLineSense = LineSense(gLine)
   inActive = ((oldLineSense == InActive) AND (oldPixelSense == InActive))
   If ((NOT inActive) AND UseConstant)
      Sample = ConstantColor
   else
      Sample = Fetch(adr)
   EndIf
   line = line + 1
   If ((LineSense(line) == "0") AND ((oldLineSense == "0"))
      adr = adr + DeltaLine
   EndIf
EndDo
```

The setting for such variables as `FirstSampleLine`, `FirstSamplePixel`, and `FirstSampleAdr` is in the address generator section that responds to carry out flags from the Kernel Address Generator, as well as AdvanceLine pulses. The logic for this part of the address generation is shown in the following pseudocode:

```
FirstSamplePixel = 0
FirstSampleLine = 0
```

```
FirstSampleAdr = FirstLineSampleAdr = ActiveStartAddress
count = 0
Do Forever
   If ((KernelXCarryOut) OR (AdvanceLine AND KernelYCarryOut) OR (count < 5))
      Do N Samples for this color plane (see pseudocode above)
   EndIf
   oldPixelSense = PixelSense(FirstSamplePixel)
   oldLineSense = LineSense(FirstSampleLine)
   If (AdvanceLine AND KernelYCarryOut)
      count = 0
      FirstSampleLine = FirstSampleLine + 1
      FirstSamplePixel = 0
      If ((LineSense(FirstSampleLine) == "0") AND (oldLineSense == "0"))
         FirstLineSampleAdr = FirstLineSampleAdr + DeltaLine
      EndIf
      FirstSampleAdr = FirstLineSampleAdr
   ElseIf (KernelXCarryOut OR (count < 5))
      FirstSamplePixel = FirstSamplePixel + 1
      count = count + 1
      If ((PixelSense(FirstSamplePixel) == "0") AND (oldPixelSense == "0"))
         FirstSampleAdr = FirstSampleAdr + DeltaPixel
      EndIf
   EndIf
EndDo
```

24.7.5.3 Register Summary

There are a number of registers that must be set before printing an image. They are summarized here in Table 35. To rotate an image by 90 degrees, simply exchange the DeltaLine and DeltaPixel values, and provide a new DeltaColumn value.

Table 35. Registers Required to be set by Caller

Register Name	Description
Image Access Parameters	
UseConstant	If 0, image edge replication occurs on reads out of image bounds. If 1, a constant color is returned.
Red	
ActiveStartAddressR	The address of red sample (ActiveStartPixel, ActiveStartLine) in ImageRAM
ActiveStartLineR	The first valid line for the image in red space (in relation to line 0)
ActiveEndLineR	The first line out of bounds for the image in red space
ActiveStartPixelR	The first valid pixel for the image in red space (in relation to pixel 0)
ActiveEndPixelR	The first pixel out of bounds for the image in red space
DeltaLineR	The amount to add to the current address to move from one line to the next in red space
DeltaPixelR	The amount to add to the current address to move from one pixel to the next on the same line in red space
DeltaColumnR	The amount to add to the current address to move from a pixel in the last line of the Active image area to the same pixel on the first line of the Active image area in red space.
ConstantColorR	Red color value to use if address out of bounds and UseConstant=1
Green	
ActiveStartAddressG	The address of green sample (ActiveStartPixel, ActiveStartLine) in ImageRAM

Table 35. Registers Required to be set by Caller

Register Name	Description
ActiveStartLineG	The first valid line for the image in green space (in relation to line 0)
ActiveEndLineG	The first line out of bounds for the image in green space
ActiveStartPixelG	The first valid pixel for the image in green space (in relation to pixel 0)
ActiveEndPixelG	The first pixel out of bounds for the image in green space
DeltaLineG	The amount to add to the current address to move from one line to the next in green space
DeltaPixelG	The amount to add to the current address to move from one pixel to the next on the same line in green space
DeltaColumnG	The amount to add to the current address to move from a pixel in the last line of the Active image area to the same pixel on the first line of the Active image area in green space.
ConstantColorG	Green color value to use if address out of bounds and UseConstant=1
Blue	
ActiveStartAddressB	The address of blue sample (ActiveStartPixel, ActiveStartLine) in ImageRAM
ActiveStartLineB	The first valid line for the image in blue space (in relation to line 0)
ActiveEndLineB	The first line out of bounds for the image in blue space
ActiveStartPixelB	The first valid pixel for the image in blue space (in relation to pixel 0)
ActiveEndPixelB	The first pixel out of bounds for the image in blue space
DeltaLineB	The amount to add to the current address to move from one line to the next in blue space
DeltaPixelB	The amount to add to the current address to move from one pixel to the next on the same line in blue space
DeltaColumnB	The amount to add to the current address to move from a pixel in the last line of the Active image area to the same pixel on the first line of the Active image area in blue space.
ConstantColorB	Blue color value to use if address out of bounds and UseConstant=1
White Balance and Range Expansion Parameters	
RedLowThreshold	8-bit value subtracted from red input values
GreenLowThreshold	8-bit value subtracted from green input values
BlueLowThreshold	8-bit value subtracted from blue input values
RedScaleFactor	8-bit scale factor used for range expansion of red pixels
GreenScaleFactor	8-bit scale factor used for range expansion of green pixels
BlueScaleFactor	8-bit scale factor used for range expansion of blue pixels

24.7.6 Resample

The Resample process is responsible for generating the full complement of R, G, and B pixels for each CFA coordinate by appropriate resampling the white-balanced and range-expanded R, G, and B planar images, as described in Section 23.2.4 on page 115.

The time allowed for producing the components of R, G, and B is 72 cycles. However we must effectively produce RGB values for 3 pixel coordinates: the pixel in question, and the pixel above and below. Thus we have 72 cycles in which to calculate 3 RGB samples.

Buffering RGB values to save recalculation requires too much memory, and in any case, we have sufficient time to generate the RGB values.

24.7.6.1 Resampling

The resampling process can be seen as 3 sets of RGB generation, each of which must be completed within 24 cycles (for a total maximum elapsed time of 72 cycles). The process of generating a single RGB value can in turn be seen as 3 processes performed in parallel: the calculation of R, the calculation of G, and the calculation of B, all for a given pixel coordinate. The theory for generating each of these values can be found in Section 23.2.4 on page 115, but the upshot is effectively running three image reconstruction filters, one on each channel of the image. In the case of the PPCIP we perform image reconstruction with 5 sample points, requiring 4 coefficients in the convolution kernel (since one coefficient is always 0 and thus the sample point is not required).

Consequently, calculation of the medium resolution R pixel is achieved by running an image reconstruction filter on the R data. Calculation of the medium resolution G pixel is achieved by running an image reconstruction filter on the G data, and calculation of the medium resolution B pixel is achieved by running an image reconstruction filter on the B data. Although the kernels are symmetric in x and y, they are not the same for each color plane. R and B are likely to be the same kernel due to their similar image characteristics, but the G plane, due to the rotation required for image reconstruction, must have a different kernel. The high level view of the process can be seen in Figure 98. Address generation is not shown.

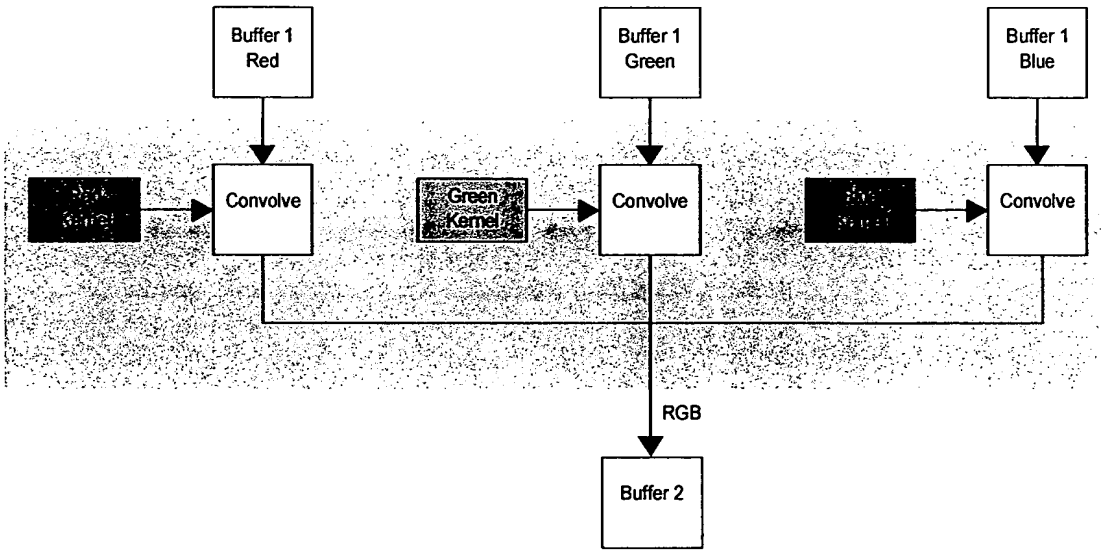


Figure 98. High Level view of Resample Process

The resampling process can only begin when there are enough pixels in Buffer 1 for the current pixel line being generated. This will be the case once 4 columns of data have been written to each of the color planes in Buffer 1. The Resampling process must stall until that time.

To calculate a given color plane's medium resolution pixel value, we have 24 cycles available. To apply the kernel to the 4×4 sample area, we apply the 1D kernel (indexed by x) on each of the 4 rows of 4 input samples. We then apply the 1D kernel (indexed by y) on the resultant 4 pixel values. The final result is the output resampled pixel. Applying a single

coefficient each cycle gives a total of 16 cycles to generate the 4 intermediate values, and 4 cycles to generate the final pixel value, for a total of 20 cycles.

With regards to precision, the input pixels are each 10 bits (8:2), and kernel coefficients are 12 bits. We keep 14 bits of precision during the 4 steps of each application of the kernel (8:6), but only save 10 bits for the result (8:2). Thus the same convolve engine can be used when convolving in x and y. The final output or R, G, or B is only 8 bits.

The heart of the resampling process is the Convolve Unit, as shown in Figure 99.

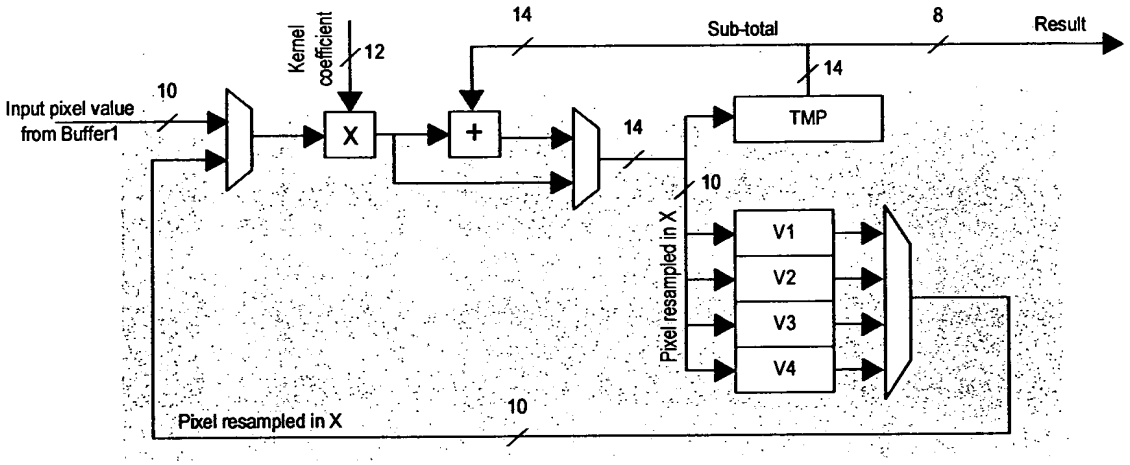


Figure 99. Structure of Convolve Unit

The process of resampling then, involves 20 cycles, as shown in Table 36. Note that the Row 1, Pixel 1 etc. refers to the input from Buffer 1, and is taken care of by the addressing mechanism (see below).

Table 36. The 20 Cycle Resample

Cycle	Kernel	Apply Kernel to:	Store Result in
1	X[1]	Row 1, Pixel 1	TMP
2	X[2]	Row 1, Pixel 2	TMP
3	X[3]	Row 1, Pixel 3	TMP
4	X[4]	Row 1, Pixel 4	TMP, V1
5	X[1]	Row 2, Pixel 1	TMP
6	X[2]	Row 2, Pixel 2	TMP
7	X[3]	Row 2, Pixel 3	TMP
8	X[4]	Row 2, Pixel 4	TMP, V2
9	X[1]	Row 3, Pixel 1	TMP
10	X[2]	Row 3, Pixel 2	TMP
11	X[3]	Row 3, Pixel 3	TMP
12	X[4]	Row 3, Pixel 4	TMP, V3
13	X[1]	Row 4, Pixel 1	TMP
14	X[2]	Row 4, Pixel 2	TMP

Table 36. The 20 Cycle Resample

Cycle	Kernel	Apply Kernel to:	Store Result in
15	X[3]	Row 4, Pixel 3	TMP
16	X[4]	Row 4, Pixel 4	TMP, V4
17	Y[1]	V1	TMP
18	Y[2]	V2	TMP
19	Y[3]	V3	TMP
20	Y[4]	V4	TMP (for output)

24.7.6.2 Address Generation for Buffer 2

The first set of RGB values is ready after 20 cycles. To reduce the number of lines to Buffer 2 we simply write 8 bits of R, G, and B to the first of the 3 sets of RGB in Buffer 2 during cycles 21, 22, and 23 respectively. During the next set of 24 cycles we spend 20 cycles calculating the values, 3 cycles transferring the R, G, and B to the second of the 3 sets of RGB in Buffer 2, and 1 idle cycle. The final set of 24 cycles is the same - calculating the RGB values in the first 20 cycles, transferring the three values over the next 3 cycles, and 1 idle cycle.

24.7.6.3 Address Generation for Kernel Lookup

The method of calculating the kernel address is the same as described at the end of Section 23.2.4 on page 115. Because we are scaling up by 2, the delta values are 1/2. Consequently each kernel is 1 dimensional, with 2 entries in the table. The most significant bit (truncated) of the fractional component in the current kernel space is used to index into the kernel coefficients table. For the first 16 cycles, the X ordinate is used to index the kernel, while in the next 4 cycles, the Y ordinate is used. Since the kernel is symmetric, the same kernel can be used for both X and Y.

For each of the 534 resampled values, we need to produce 3 pixels - the pixel in question, and the pixels above and below that pixel. Rather than generate a center pixel and then move up and down from that center pixel, we generate a pixel and generate the two pixels below it. The second pixel generated is taken to be the center pixel. We then return to the original row and generate the next 3 pixels in the next output position. In this way, as shown in Figure 100, we generate 3 pixels for each of the 534 positions.

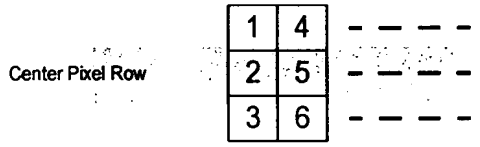


Figure 100. Order of Pixels being Generated

Thus we have a current position in kernel space. As we advance to the next pixel in X or Y in original input space, we add appropriate delta values to these kernel coordinates. Looking at Figure 101, we see the two cases for rotated and unrotated input space.

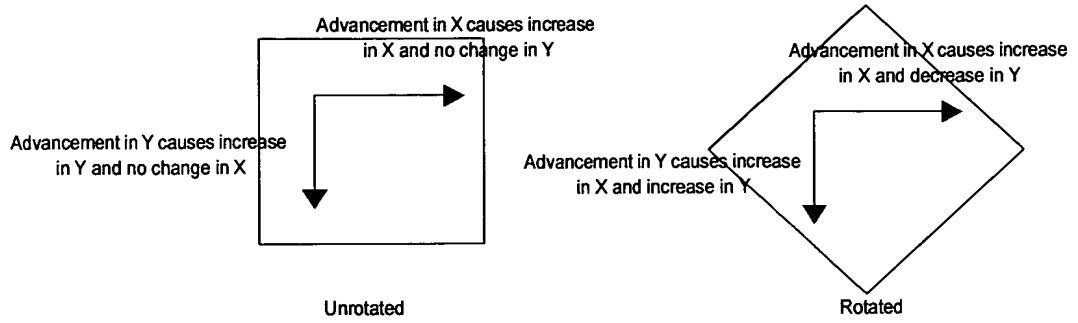


Figure 101. Movement in X or Y in Rotated Space

We consider the movement in X and Y as ΔX and ΔY, with their values dependent on the value of ops (see Section 23.2.4 on page 115). For the green channel, $\Delta X = \Delta Y = 1/2$. For the red and blue channels, $\Delta X = 1/2$ and $\Delta Y = 0$.

We can now apply the ΔX and ΔY values to movement within the kernel. Consequently, when we advance in X, we add ΔX to X and subtract ΔY from Y. In the unrotated case, this merely subtracts 0 from Y. Likewise, when we advance in Y, we add ΔY to X and ΔX to Y. We can do this because movement in X and Y differs by 90 degrees.

The address generation for kernel lookup assumes a starting position set by software, and two deltas ΔX and ΔY with respect to movement in Y in kernel space. The address generation logic is shown in the following pseudocode:

```
ColumnKernelY = StartKernelY
ColumnKernelX = StartKernelX
Do 850 times (however many output lines there are to process)
   KernelX = ColumnKernelX
   KernelY = ColumnKernelY
   Do 534 times
      GeneratePixel
      KernelX = KernelX + DeltaY (movement in Y)
      KernelY = KernelY + DeltaX (movement in Y)
      Generate Pixel
      KernelX = KernelX + DeltaY (movement in Y)
      KernelY = KernelY + DeltaX (movement in Y)
      GeneratePixel
      KernelX = ColumnKernelX + DeltaX (movement in X)
      KernelY = ColumnKernelY - DeltaY (movement in X)
   EndDo
   ColumnKernelY = ColumnKernelY + DeltaX (movement in Y)
   ColumnKernelX = ColumnKernelX + DeltaY (movement in Y)
EndDo
```

As shown in the pseudocode, the generation of 3 pixels occurs 534 times. Associated with the generation of each pixel is 2 additions, which can be performed during the course of the `GeneratePixel` 24 cycle task. Each 24 cycle `GeneratePixel` task consists of 4 sets

of 4 cycles indexing the kernel via KernelX (coefficients 0, 1, 2, 3), followed by 4 cycles indexing the kernel via KernelY (coefficients 0, 1, 2, 3), followed by 4 wait cycles.

Note that all values are positive and fractional only. The two carry outs from the updating of the X and Y kernel values are output to the address generation of Buffer 1 (see Section 24.7.6.4 on page 143 below). These carry out flags simply indicate whether or not the particular ordinates for the kernel wrapped during the mathematical operation. Wrapping can be either above 1 or below 0, but the result is always positive.

The two carry out bits are also sent to the WhiteBalance/RangeExpansion Unit for use in determining the relative input lines from the image.

24.7.6.4 Address Generation for Buffer 1

The Resampler reads from Buffer 1, which consists of 3 individually addressable buffers - one for each color plane. Each buffer can either be read from or written to during each cycle.

The reading process of 72 cycles is broken down into 3 sets of 24 cycles, one set of 24 cycles for the generation of each pixel. Each 24 cycle set involves 16 reads from Buffer 1 followed by 8 cycles with no access. Buffer 1 is written to during these 8 cycles. The 16 reads from Buffer 1 are effectively 4 sets of 4 reads, and coincide with 4 groups of 4 reads to the kernel for each color plane.

The address generation then, involves generating 16 addresses for calculating the first pixel (followed by 8 wait cycles), generating 16 addresses for calculating the second pixel (followed by 8 wait cycles), and finally generating the 16 addresses for the third pixel (followed by 8 wait cycles).

Each color plane has its own starting Buffer 1 address parameters. As the 3 sets of 16 addresses are generated for each of the 534 positions along the line, and as the sampler advances from one line of 534 samples to the next, the two carry out bits from the Kernel Address Generation Unit are used to update these Buffer 1 address parameters.

24.7.6.5 Green buffer

Address generation for the green sub-buffer within Buffer 1 is more complicated than the red and blue sub-buffers for two main reasons:

- the green channel represents a checkerboard pattern in the CFA. Alternate lines consist of odd or even pixels only. To resample the green channel, we must effectively rotate the channel by 45 degrees.
- there are twice as many green pixels than red or blue pixels. Resampling means the reading of more samples in the same amount of time - there are still 16 samples read to generate each pixel, but there is a higher likelihood of advancing the buffer each time. The exact likelihood depends on the scale factor used.

However, the same concept of using a RAM as a cyclical buffer is used for the green channel. The green sub-buffer is a 66 entry RAM with a logical arrangement of 11 rows, each

containing 6 entries. The relationship between RAM address and logical position is shown in Figure 102.

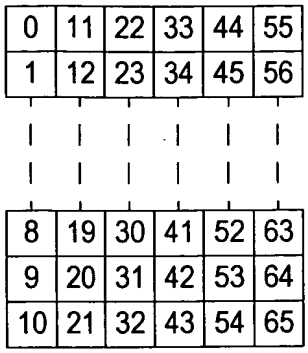


Figure 102. Address of Entries in Buffer 1's Green Sub-buffer

The samples in Buffer 1 represent a checkerboard pattern in the CFA. Consequently, samples in one row (e.g. addresses 0, 11, 22, 33, 44, 55) may represent odd or even pixels, depending on the current line within the entire image. This is illustrated in Figure 103.

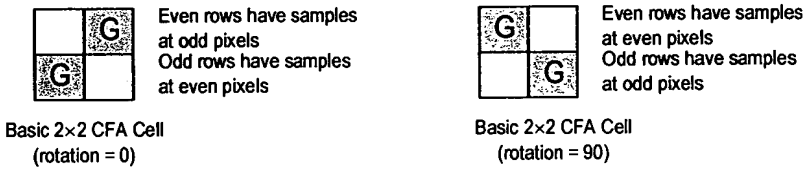


Figure 103. Samples in Each Row are for Odd or Even Pixels Depending on Rotation

Consequently, when we map a 4×4 sampling area onto the buffer, there are two possibilities for the interpretation of the samples. As a result there are two types of addressing, depending on whether the current line is represented by odd or even pixels. This means that even rows with image rotation 0 will have the same addressing as odd rows with image rotation 90 since they both hold odd pixels. Likewise, the odd rows with image rotation 0 will have the same addressing as even rows with image rotation 90 since they both hold even pixels. This means the physical CFA orientation can be taken account of. The decision is summarized in Table 37.

Table 37. Determining Sampling Type

Rotation	Current Line		Pixels	Type
0	Even Line	→	Odd	Type 2
0	Odd Line	→	Even	Type 1
90	Even Line	→	Even	Type 1
90	Odd Line	→	Odd	Type 2

The actual 4×4 sampling window is the way we effectively rotate the buffer by 45 degrees. The 45 degree rotation is necessary for effective resampling, as described in Section 23.2.4 on page 115.

Assuming for the moment that we only need to generate a single resample, we consider the buffer addressing by examining the two types of 4×4 sampling windows as shown in Figure 104.

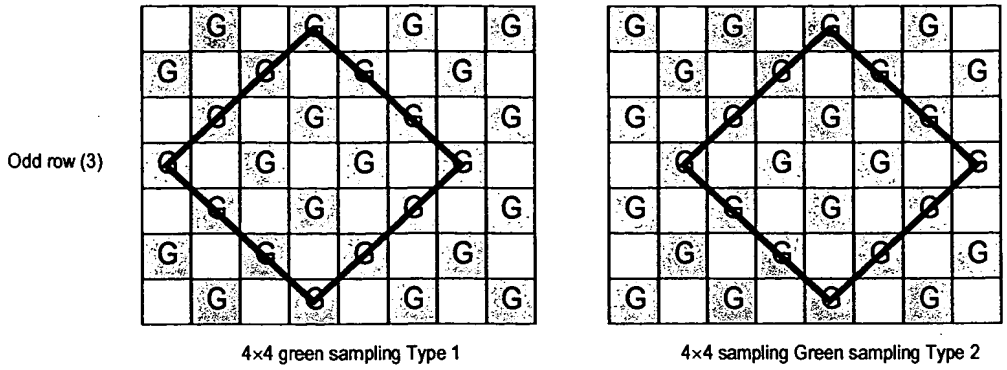


Figure 104. 4×4 Sampling of Green Channel

Although the two 4×4 sampling types look similar, the difference comes from the way in which the 4×4 mapping is represented in the planar image. Figure 105 illustrates the mapping of the Type 1 4×4 sampling to the green sub-buffer. Only the top 7 rows and rightmost 4 columns are shown since the 4×4 sample area is contained wholly within this area.

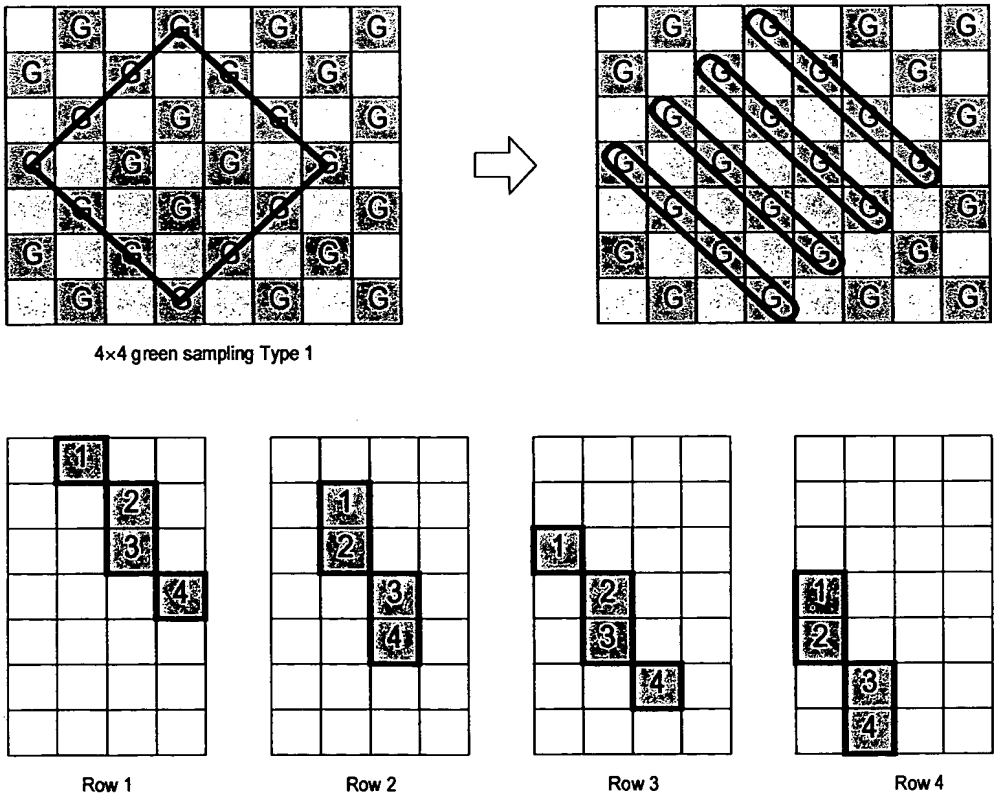


Figure 105. 4×4 Green Sampling Type 1

The mapping of buffer pixels to sample rows for the Type 2 sampling process is very similar, and can be seen in Figure 106.

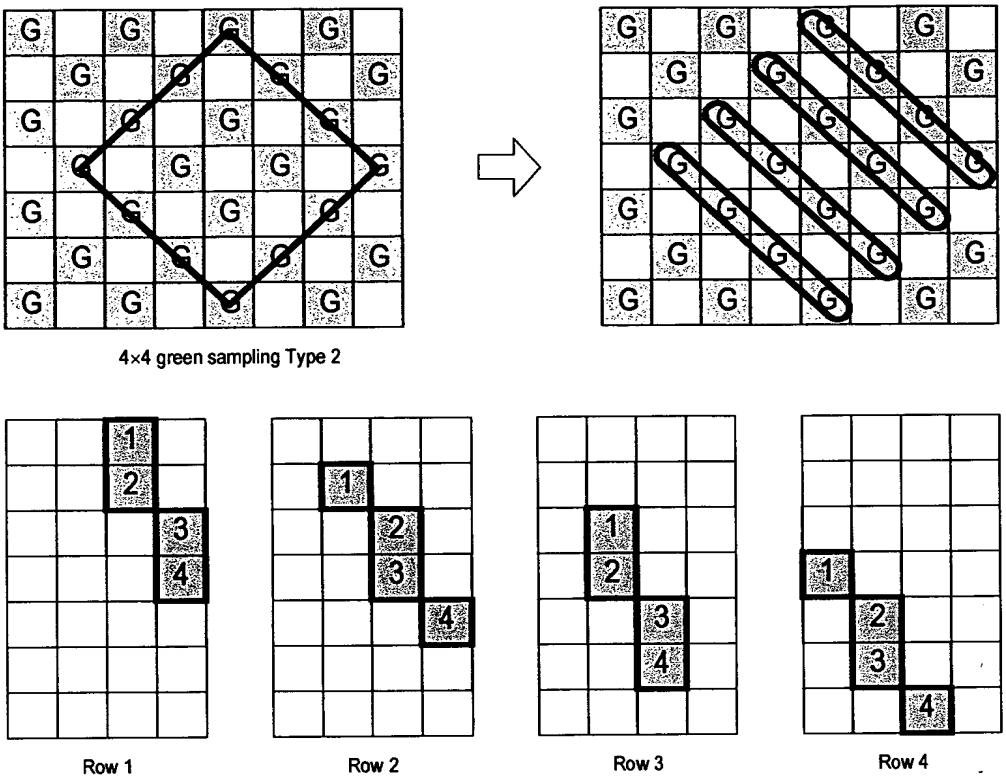


Figure 106. 4×4 Green Sampling Type 2

In both Type 1 and Type 2 addressing of the 16 samples there are two ways of processing a row. Processing of Rows 1 and 3 of Type 1 addressing is the same (relatively speaking) as processing rows 2 and 3 of Type 2. Likewise, processing rows 2 and 4 of Type 1 is the same (relatively speaking) as processing rows 1 and 3 of Type 2. We will call these row addressing methods Types A and B, as shown in Figure 107.

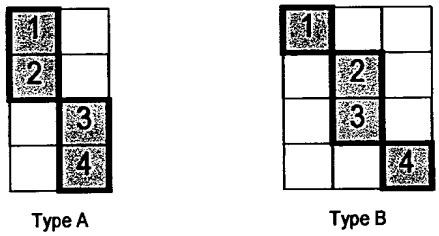


Figure 107. Two Types of Row Addressing for Green

Given a starting position for the 4×4 window (`WindowStartAdr`) and a starting type (`WindowStartType`), we can generate the addresses for the 16 samples by means of an 8 entry table (for traversing the two sets of 4 samples). When we read the first sample value we add an offset from the table to arrive at the next sample position. The offset will depend on the type (A, B = 0, 1). The offset from the fourth sample is the amount needed to arrive at the first sample point for the next line (and must take account of the number of

sample columns). After generating each row of 4 samples, we swap between TypeA and TypeB. The logic for generating the addresses for a single set of 16 samples is shown in the following pseudocode. The addition modulo 66 caters for the cyclical buffer.

```
Adr = WindowStartAdr
TypeAB = WindowStartType
Do 4 times
   For N = 0 to 4
      Fetch Adr
      Adr = (Adr + Table[TypeAB,N]) mod 66
   EndFor
   TypeAB = NOT TypeAB
EndDo
```

The lookup table consists of 8 entries: 4 for Type A, and 4 for Type B address offset generation. The offsets are all relative to the current sample position (`Adr`).

Table 38. Offset Values for 16-Sample Address Generation

TypeAB	N	Offset
0	0	12
0	1	1
0	2	12
0	3	31
1	0	1
1	1	12
1	2	1
1	3	31

At the end of the 16 reads, the `TypeAB` bit will be the same as the original value (loaded from `WindowStartType`).

Reading a single set of 16 samples is not enough. Three sets of 16 samples must be read (representing 3 different positions in Y in unrotated input space). At the end of the first and second set of 16 samples, the kernel positions are updated by the kernel address generator. The carry bits from this update are used to set the window for the next set of 16 samples. The two carry bits index into a table containing an offset and a 1-bit flag. The offset is added to the `WindowStartAdr`, and the flag is used to determine whether or not to invert `WindowStartType`. The values for the table are shown in Table 39.

Table 39. Updating WindowStartAdr and WindowStartType

KernelX CarryOut	KernelY CarryOut	Offset	Type
0	0	0	No change
0	1	1	Invert
1	0	12	Invert
1	1	2	No change

At the end of the third set of 16 samples, the kernel positions are updated to compensate for advancement in X in unrotated input space. This time, a different motion direction is

produced, so a different Offset/TypeAB modifying table is used. We cannot add these offsets to the current WindowStartAdr value, because that represents a position two movements in Y away from where we want to start the movement. Consequently we load WindowStartAdr and WindowStartType from another set of variables: TopStartAdr and TopStartAdr, representing the first entry in the current line of 534. The two carry out flags from the Kernel address generator are used to lookup Table 40 to determine the offset to add to TopStartAdr and whether or not to invert TopStartType. As before, the addition is modulo 66 (the size of the green RAM). The results are copied to WindowStartAdr and WindowStartType for use in generating the next 3 sets of 16 samples.

Table 40. Updating TopStartAdr and TopStartType

KernelX CarryOut	KernelY CarryOut	Offset	Type
0	0	0	No change
0	1	10	Invert
1	0	12	Invert
1	1	11	No change

After processing the 534 sets of 3 sets of 16 samples, the next line of 534 begins. However the address of the first sample for position 0 within the next line must be determined. Since the samples are always loaded into the correct places in Buffer 1, we can always start from exactly the same position in Buffer 1 (i.e. TopStartAdr can be loaded from a constant Position0Adr). However, we must worry about which type we are dealing with, since the type depends on how much we advanced. Consequently we have an initial Position0Type which must be updated depending on the carry out flags from the kernel address generator. Since we are moving in unrotated Y input space, the logic used is the same as for updating WindowStartType, except that it is performed on Position0Type instead. The new value for Position0Type is copied into TopStartType, and WindowStartAdr to begin sampling of the first position of the new line.

The sampling process for a given 534 position line cannot begin until there are enough entries in Buffer 1, placed there by the WhiteBalance/RangeExpansion Unit. This will occur 128 cycles after the start of each new line (see Section 24.7.5 on page 133).

24.7.6.6 Red and Blue buffers

Buffer 1's red and blue sub-buffer's are simply 2 RAMs accessed as cyclical buffers. Each buffer is 30 bytes, but has a logical arrangement of 6 rows, each containing 6 entries. The relationship between RAM address and logical position is shown in Figure 108.

0	6	12	18	24	30
1	7	13	19	25	31
2	8	14	20	26	32
3	9	15	21	27	33
4	10	16	22	28	34
5	11	17	23	29	35

Figure 108. Address of Entries in Buffer 1's Red and Blue Sub-buffers

For red and blue, the first 16 samples to be read are always the top 4×4 entries. The remaining two columns of samples are not accessed by the reading algorithm at this stage.

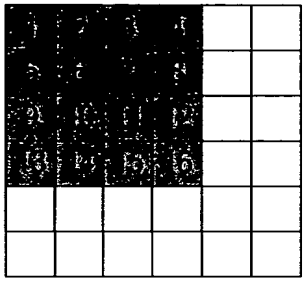


Figure 109. 16 Samples Read for Calculating First Pixel

The address generation for these first 16 samples is simply a starting position (in this case 0) followed by 16 steps of addition modulo 36, as shown in the following pseudocode:

```
ADR = StartADR
Do 4 times
   Do 4 times
      ADR = ADR + 6 MOD 36
   End Do
   ADR = ADR + 13 MOD 36
End Do
```

However, this address generation mechanism is different from the green channel. Rather than design two addressing mechanisms, it is possible to apply the green addressing scheme to the red and blue channels, and simply use different values in the tables. This reduces design complexity. The only difference then, becomes the addition modulo 36, instead of addition modulo 66. This can be catered for by a simple multiplexor.

Looking at the various address generation tables for green, and considering them as applied to red and blue, it is apparent that there is no requirement for a Type, since both the red and the blue channels do not need to be rotated 45 degrees. So that we can safely ignore the Type value, the red/blue equivalent of Table 38, shown in Table 41, has two sets of identical 4 entries.

Table 41. Offset Values for 16-Sample Address Generation (Red/Blue)

TypeAB	N	Offset
0	0	6
0	1	6
0	2	6
0	3	13
1	0	6
1	1	6
1	2	6
1	3	13

As with green address generation, we move twice in Y before advancing to the next entry of 534. For red and blue there is no scaling between movement in kernel space and movement in the input space. There is also no rotation. As we move in Y, the ΔY of 0 is added to `KernelX` (see Section 24.7.6.3 on page 141). As a result, the carry out from `KernelX` will never be set. Looking at Table 39, the only possible occurrences are `KernelX/KernelY` values of 00 or 01. In the case of 00, the green solution is no change to either WindowStartAdr or WindowStartType, so this is correct for red and blue also. In the case of 01, we want to add 1 to WindowStartAdr, and don't care about WindowStartType. The green values can therefore be safely used for red and blue. The worst case is advancement by 1 in address both times, resulting in an overlapping worst case as shown in Figure 110.

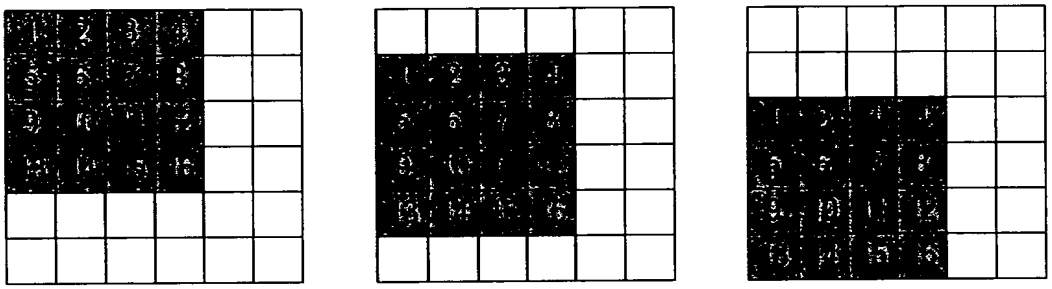


Figure 110. Overlapping Worst Case 4×4 Reading from Blue and Red Buffers

At the end of the third set of 16 samples, `TopStartAdr` and `TopStartType` must be updated. Since we are moving in X (and adding ΔY=0 to `KernelY`), the carry out from `KernelY` will always be 0. The red/blue equivalent of Table 40 is shown here in Table 42. Note that there is no `Type` column, since `Type` is not important for Red or Blue.

Table 42. Updating TopStartAdr and TopStartType (Red/Blue)

KernelX CarryOut	KernelY CarryOut	Offset
0	0	0
0	1	-
1	0	6
1	1	-

The process of advancing from one line of 534 sets of 3 pixels to the next is the same as for green. The `Position0Adr` will be the same for the first set of 16 samples for a given line (`Position0Adr` = 0 for red and blue), and `Type` is irrelevant. Generation of the next line cannot begin until there are enough samples in Buffer 1. Red and blue generation must start at the same time as green generation, so cannot begin until 128 cycles after the start of a new line (see Section 24.7.5 on page 133).

24.7.7 Convert to L*a*b*

The conversion from RGB to L*a*b* is performed as described in Section 23.2.5 on page 119.

The color conversion process must produce contone L*a*b* pixels for the Sharpen process within 72 cycles. Since the sharpening process only requires the L* values corresponding to the first and third RGB sets, and only requires the full L*a*b* set for the second RGB set, we have 72 cycles in which to perform 5 color conversions (3 sets of RGB to L*, and 1 set each of RGB to a* and RGB to b*).

The process as described here requires 14 cycles per color component, leading to a total of 70 cycles for 5 conversions (leaving 2 cycles spare).

The conversion is performed as tri-linear interpolation. Three 17 × 17 × 17 × 8-bit lookup tables are used for the conversion process: RGB to L*, RGB to a*, and RGB to b*.

Address generation for indexing into the lookup tables is straightforward. We use the 4 most significant bits of each 8-bit color component for address generation, and the 4 least significant bits of each 8-bit color component for interpolating between values retrieved from the conversion tables. The addressing into the lookup table requires an adder due to the fact that the lookup table has dimensions of 17 rather than 16. Fortunately, multiplying a 4-bit number X by 17 is an 8-bit number XX, and therefore does not require an adder or multiplier, and multiplying a 4 bit number by 17^2 (289) is only slightly more complicated, requiring a single add.

Although the interpolation could be performed faster, we use a single adder to generate addresses and have a single cycle interpolation unit. Consequently we are able to calculate the interpolation for generating a single color component from RGB in 14 cycles, as shown in Table 43. The process must be repeated 5 times, once for each color conversion. Faster methods are possible, but not necessary.

Table 43. Trilinear interpolation for color conversion

Cycle	Load	Effective Fetch	Adjust ADR register	Interpolate
1			ADR = 289R	
2			ADR = ADR +17G	
3			ADR = ADR + B	
4	P1	RGB	ADR = ADR + 1	
5	P2	RGB+1	ADR = ADR + 16	
6	P1	RG+1B	ADR = ADR + 1	P3 = P1 to P2 by B
7	P2	RG+1B+1	ADR = ADR + 271	
8	P1	R+1GB	ADR = ADR + 1	P4 = P1 to P2 by B
9	P2	R+1GB+1	ADR = ADR + 16	P5 = P3 to P4 by G
10	P1	R+1G+1B	ADR = ADR + 1	P3 = P1 to P2 by B
11	P2	R+1G+1B+1		
12				P4 = P1 to P2 by B
13				P6 = P3 to P4 by G
14				V = P5 to P6 by R

As shown in Table 43, a single ADR register and adder can be used for address generation into the lookup tables. 6 sets of 8-bit registers can be used to hold intermediate results - 2 registers hold values loaded from the lookup tables, and 4 registers are used for the output from the interpolation unit. Note that the input to the linear interpolation unit is always a pair of 8-bit registers P1/P2, P3/P4, and P5/P6. This is done deliberately to reduce register

selection logic. In cycle 14, the "V" register holds the 10-bit value finally calculated. The 8-bit result can be written to the appropriate location in Buffer 3 during the next cycle.

A block diagram of the color conversion process can be seen in Figure 111.

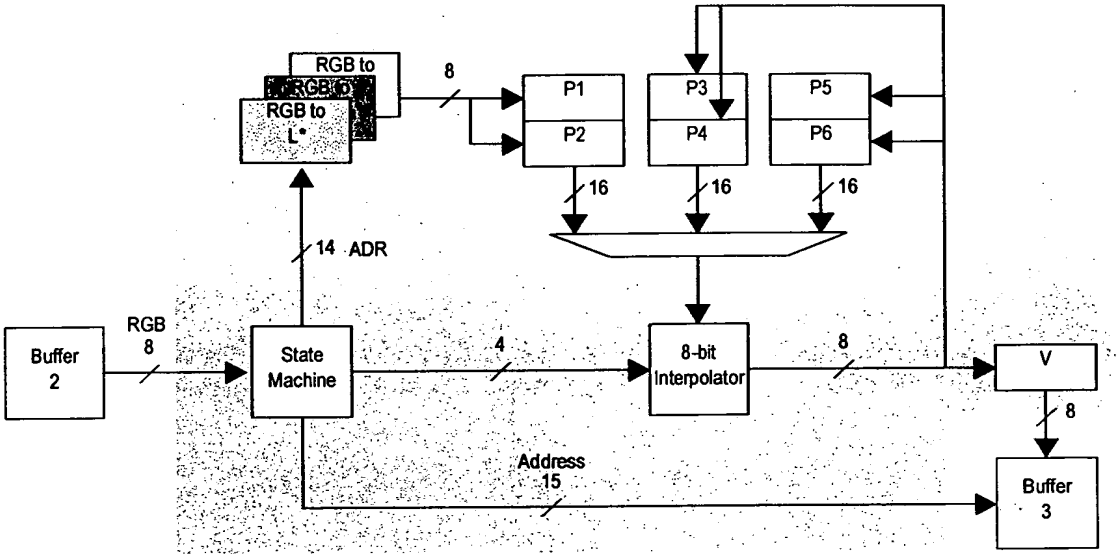


Figure 111. Convert from RGB to L*a*b*

The basic 14 cycle color conversion process is run 5 times as follows:

- on RGB_1 to generate L^*_1
- on RGB_2 to generate L^*_2
- on RGB_3 to generate L^*_3
- on RGB_2 to generate a*
- on RGB_2 to generate b*

Address generation for writing to Buffer 3 makes use of the cyclical nature of Buffer 3. The address consists of a 2-bit column component (representing which of the 4 columns should be written to), and a 3-bit value representing L^*_1, L^*_2, L^*_3, a*, or b*. The column number starts at 0 each new line and increments (with wrapping) every 72 cycles. The order of writing to Buffer 3 is shown in Table 44. The C register is the 2-bit column component of the address. All addition on C is modulo 4 (wraps within 2 bits).

Table 44. Write Access to Buffer 3 during 72 Cycle set

Cycle	Address	Update C
0		C=C+1
14	C, L1	
28	C, L2	
42	C, L3	
56	C, a*	
70	C, b*	

24.7.8 Sharpen

The Sharpen Unit performs the sharpening task described in Section 23.2.6 on page 120. The Sharpen Unit must keep up with the PenPrint Serial Bus transmission speed, which implies a complete pixel must be sharpened within 72 cycles.

The sharpening process involves a highpass filter of the luminance channel of the image (the L* channel). The highpass filter used is a basic highpass filter using a 3×3 convolution kernel, as shown in Figure 112.

$$\frac{1}{9} \times \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Figure 112. A basic highpass spatial filter

The high pass filter is calculated over 10 cycles. The first cycle loads the temporary register with 8 times the center pixel value (the center pixel shifted left by 3 bits). The next 8 cycles subtract the remaining 8 pixel values, with a floor of 0. Thus the entire procedure can be accomplished by an adder. Cycle 10 involves the multiplication of the result by a constant. This constant is the representation of 1/9, but is a register to allow the amount to altered by software by some scale factor.

The resultant sharpened L* is written out to Buffer 4 during cycle 11, and the a*, and b* color components are copied to Buffer 4 during cycles 12 and 13.

The structure of the Sharpen unit can be seen in Figure 113.

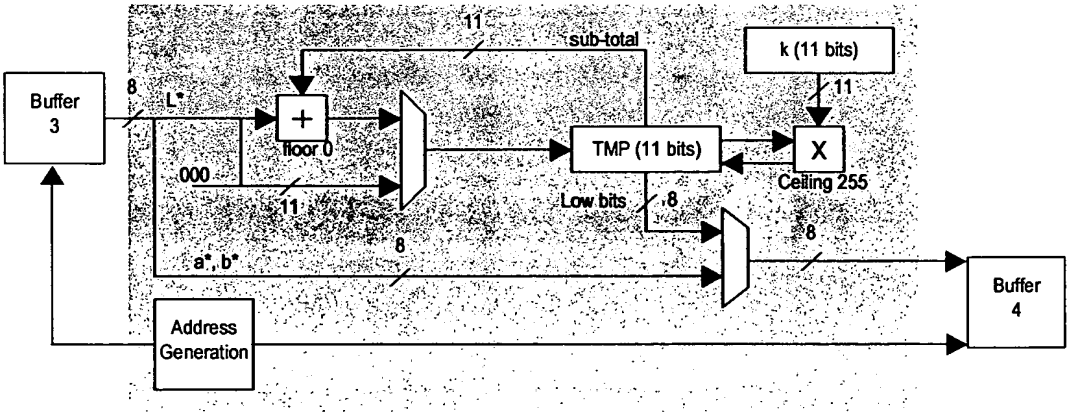


Figure 113. Structure of the Sharpen Unit

The adder unit connected to Buffer 2 is a subtractor with a floor of 0. TMP is loaded with 8 × the first L value during cycle 0 (of 75), and then the next 8 L values are subtracted from it. The result is not signed, since the subtraction has a floor of 0.

During the 10th cycle (Cycle 9), the 11 bit total in TMP is multiplied by a scale factor (typically 1/9, but under software control so that the factor can be adjusted) and written back to TMP. Only 8 integer bits of the result are written to TMP (the fraction is truncated), so the limit from the multiply unit is 255. If a scale factor of 1/9 is used, the maxi-

mum value written will be 226 (255 × 8 / 9). The scale factor is 8 bits of fraction, with the high bit representing 1/8.

Address Generation is straightforward. Writing to Buffer 4 is simply L*, a*, and b* in cycles 11, 12, and 13 respectively. Reading from Buffer 3 makes use of the cyclical nature of Buffer 3. The address consists of a 2-bit column component (representing which of the 4 columns should be read), and a 3-bit value representing L^*_1, L^*_2, L^*_3, a*, or b*. The column number starts at 1 each line and increments (with wrapping) every 72 cycles. The order of reading Buffer 3 is shown in Table 45. The C register is the 2-bit column component of the address. All addition on C is modulo 4 (wraps within 2 bits).

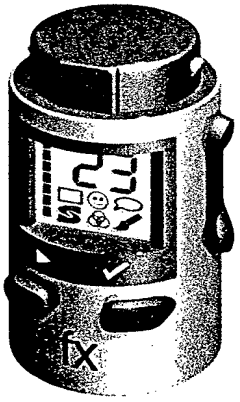
Table 45. Read Access to Buffer 3 during 72 Cycle set

Cycle	Address	Update C
0	C, L2	C=C-1
1	C, L1	
2	C, L2	
3	C, L3	C=C+1
4	C, L1	
5	C, L3	C=C+1
6	C, L1	
7	C, L2	
8	C, L3	C=C-1
9-10	No access	
12	C, a*	
13	C, b*	C=C-1
14-71	No access	

After Cycle 13, the C register holds the column number for the next calculation set, thus making the fetch during the next Cycle 0 valid.

Sharpening can only begin when there have been sufficient L*a*b* pixels written to Buffer 3 (so that the highpass filter is valid). The sharpen process must therefore stall until the color conversion process has written 3 columns of data to Buffer 3.

Effects Module



25 Effects Module

25.1 OVERVIEW

The PenPrint Effects Module is an image processing module. It allows a user to select a number of effects and applies them to the current image in the PenPrint Printer Module. The effects include borders, clip-art, captions, warps, color changes, and painting styles.

The PenPrint Effects Module obtains power from the PenPrint Serial Bus.

25.2 APPEARANCE

Figure 114 shows the PenPrint Effects Module. Note that the LCD panel is showing all segments active.

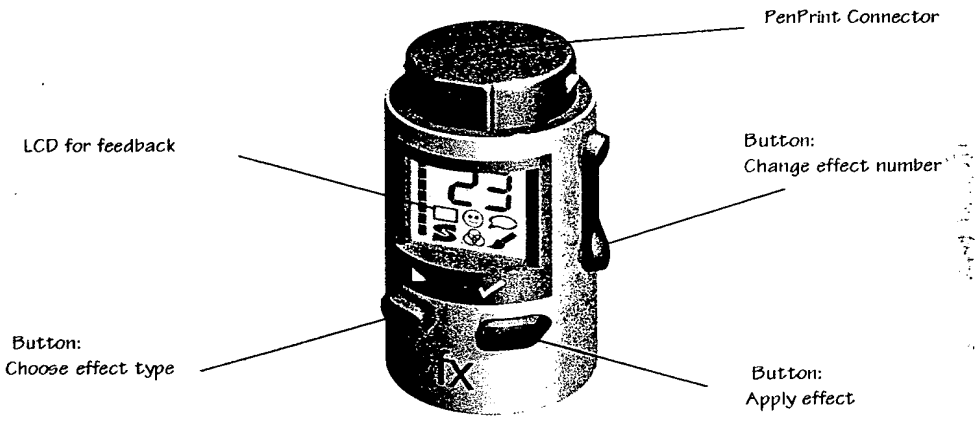


Figure 114. PenPrint Effects Module Showing all LCD Segments

25.3 METHOD OF OPERATION

The basic level of operation is to select an effect and then to apply that effect to the current image in the PenPrint Printer Module.

The LCD provides feedback in the form of an effect type and number, and an animated thermometer that is shown while the effect is being applied to the image.

Pressing the Effect Type button cycles through the effect types. Each effect type has its own LCD icon and a current effect number. The effect types are borders, clip-art, captions, warps, color changes, and painting styles.

Pressing the side up/down button increments or decrements the effect number. The effect number is visible on the LCD screen. Ideally each effect type has 100 effects (although this is not strictly required). The current effect number for each effect type is remembered by the Effect Module so that the next time the effect type is chosen, the previously used effect number is selected.

Once the desired effect type and effect number have been chosen, the effect can be applied to the image currently in the PenPrint Printer Module. This is done by pressing the Apply

button below the LCD. As the image is being read, the effect applied, and resultant image written back, a series of animated segments appears on the left side of the LCD to form a thermometer-style status bar. The proportion of segments displayed is the proportion of the work that has been done so far. The user knows that the effect has been applied once all the segments have been displayed.

The user must consult a manual to determine the correspondence between effect and effect number.

To apply multiple effects to an image a user simply applies each effect one at a time. The order may be important if a specific end image is desired. For example, adding a border and then warping the image will produce a different result to warping the image followed by adding a border.

25.4 EFFECT TYPES

The Effects module provides 6 basic effect types:

- borders
- clip-art
- captions
- warps
- color changes
- painting styles

These effects can be combined in any way and in any order. Figure 115 shows two life size samples of printouts from the Effects Module.

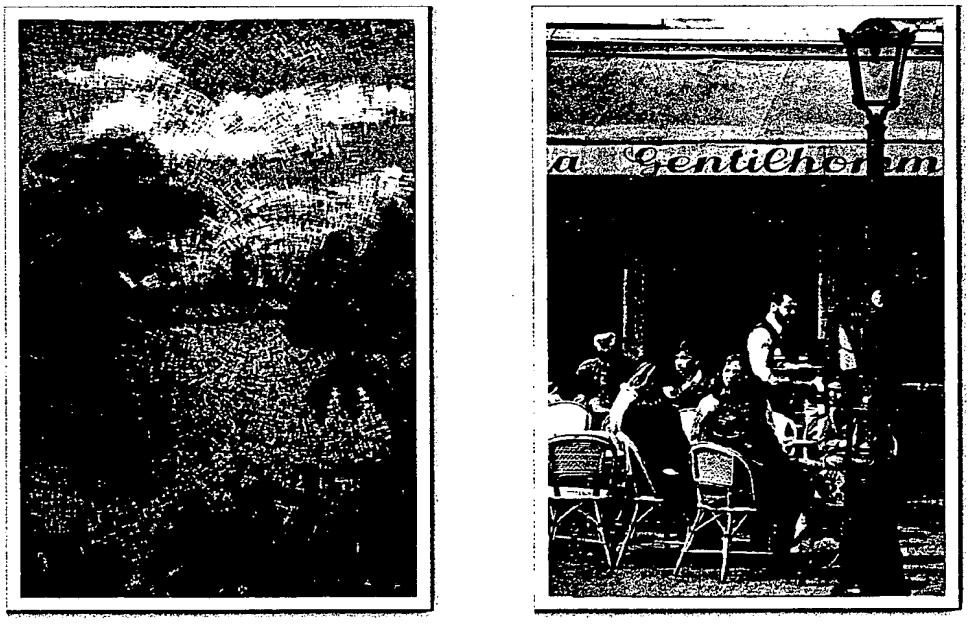


Figure 115. Sample Images with Effects Applied (life size)

25.4.1 Borders

Image borders can range from simple white to elaborate and complex designs. Some borders incorporate the image, but the majority of borders and frames are straightforward, consisting of one image being overlaid over another. Figure 116 shows some life sized examples of border effects.

Figure 116. Sample Border Effects (life size)

25.4.2 Clip-art

Clip-art consist of photographic and stylized cartoon characters and images inserted into the image at a specific location. The characters and clip-art included in a general Effects Module would require little or no licensing (compared to the Kid's Character Module described below). Examples include clowns, action figures, smiley faces, ugly faces, animals, specific objects etc.

Note that no effort is made to interpret the image before the character is added. For a given effect number a specific character is added at a specific fixed place in the image.

The Character Module, by contrast, only contains Clip-art effects of a given topic or genre. Examples include The Simpsons, Star Wars, Batman, and Dilbert as well as company specific modules for McDonalds etc. See "Character Module" on page 46 for more information. Figure 117 shows some life sized examples of clip-art effects.

Figure 117. Sample Clip-art Effects (life size)

25.4.3 Captions

Captions consist of speech bubbles, lines of text or other textual effects applied to an image. Examples include "Happy Birthday!", "How old are you?", "It was *this* big!", "Wow!", "Help!" etc.

Note that no effort is made to interpret the image before the caption is added. For a given effect number a specific caption is added at a specific fixed place in the image.

Captions are overlaid as graphics. Therefore they can include anything and be in any language, any color combination, and be in multiple fonts. The user is unable to change any part of the caption or style - the caption is simply added to the image. Choice is available due to the 100 caption numbers. Figure 118 shows some life sized examples of caption effects.

Figure 118. Sample Caption Effects (life size)

25.4.4 Warps

Images can be warped using the Warp effect type. Each warp effect number affects a fixed area of the image, which can be the entire image, or a limited part of the image. As with other effect modes, no effort is made to interpret the image before the warp is applied. For a given effect number, the specific warp is added at a specific fixed place in the image.

Some warps are specifically designed to affect a face image, although there is no specific face detection mechanism. Instead, the warp is applied to the part of the image where the face *should* be. Of course there may not be a face there at all, so the warp will only be applied to that fixed area of the image. Figure 119 shows some life sized examples of warp effects.

Figure 119. Sample Warp Effects (life size)

25.4.5 Color Changes

It is often desirable to transform an image in terms of color. Simple color effects include removal of color to produce a greyscale image or a sepia tone image. More complex effects include exaggeration of certain colors, substitution of one color for another etc. Figure 120 shows some life sized examples of color effects.

Figure 120. Sample Color Effects (life size)

25.4.6 Painting Styles

The painting styles effect mode lets the user apply a painting style to an image. For example, an image can be changed into an impressionist drawing of the original, or drawn with a variety of brush strokes from a limited color palette. Painting styles also includes limited tiling effects.

It should be noted that these painting styles do not utilize lighting calculations. Certain types of metallic or bump-map based painting effects are therefore not possible. Figure 121 shows some life sized examples of painting effects.

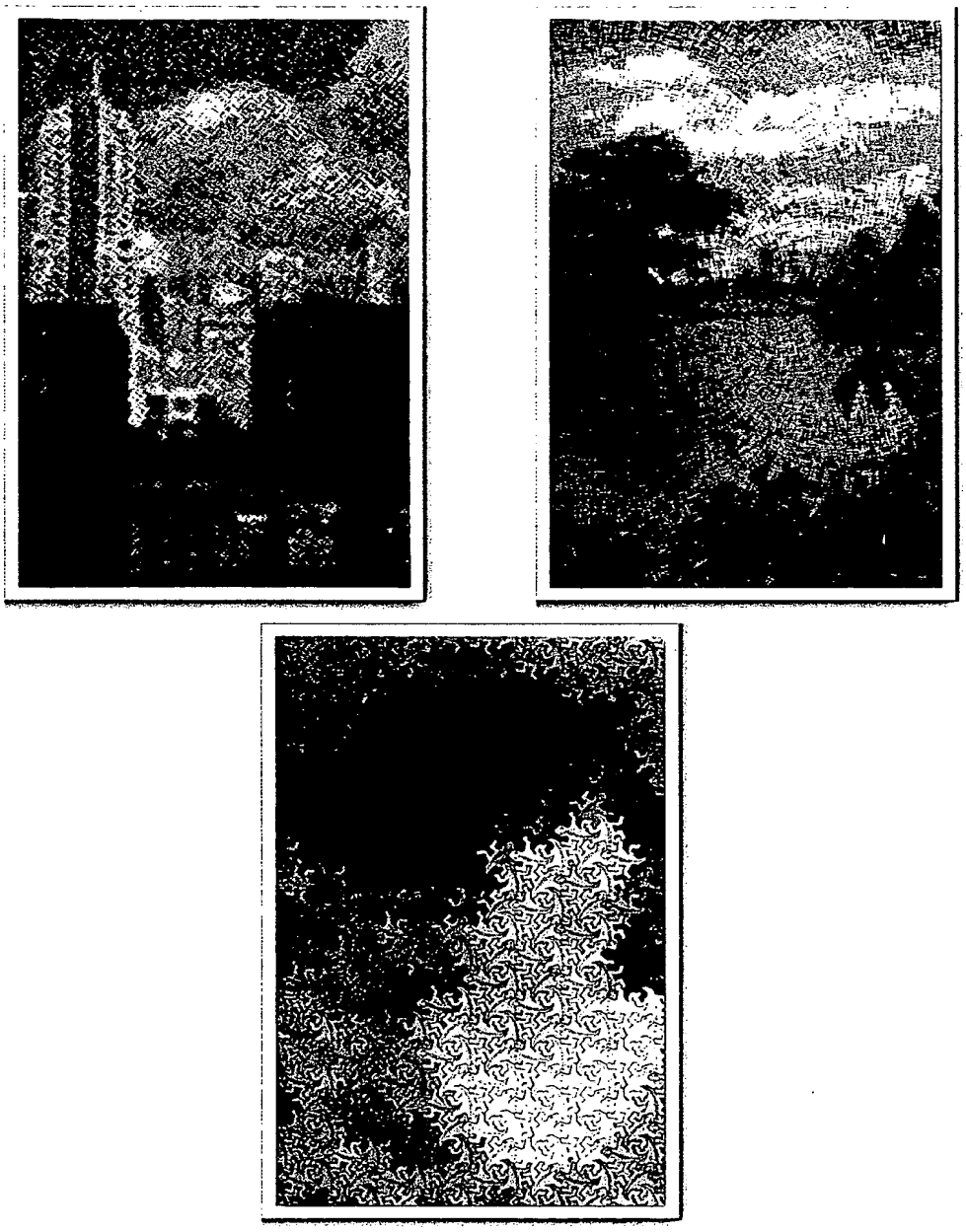


Figure 121. Sample Painting Effects (life size)

25.5 INTERNALS

The Effects Module is built with:

- a standard PenPrint male bayonet connector
- a standard PenPrint male bayonet connector
- a special-purpose LCD
- 2 single buttons and 1 double button
- an Effects Module Central Processor (an ASIC containing a CPU, DSP, and an implementation of the VARK image processing language [7] for producing the effects)
- a ROM for program and data

An exploded view of the Effects Module can be found in Figure 122.

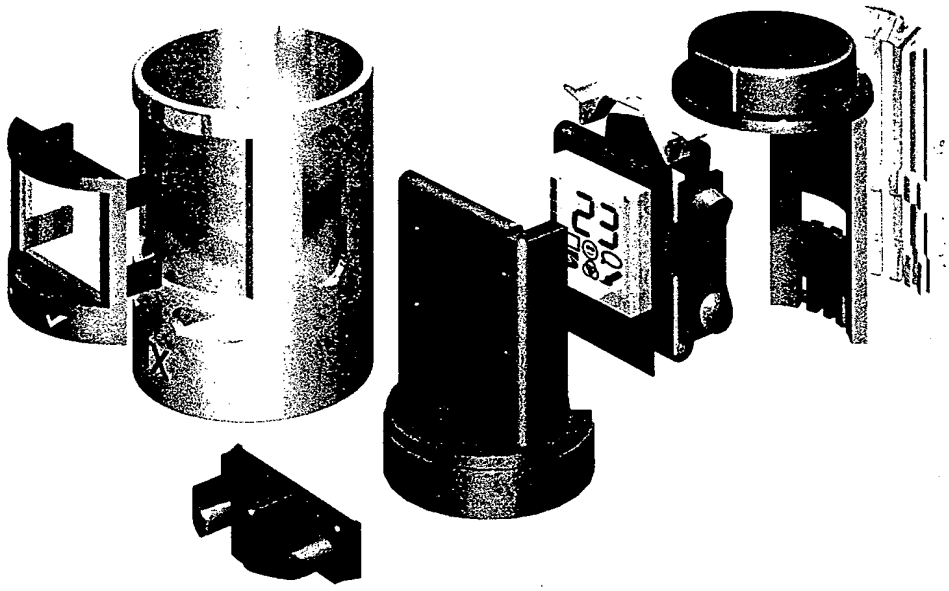


Figure 122. Exploded View of Effects Module

26 Effects Module Central Processor

The Effects Module Central Processor (EMCP) is a single chip containing a standard RISC processor core with DSP for fast integer multiplication and a small memory. The software running on the EMCP is a limited subset of the VARK language [7] and a lookup table of VARK scripts.

Since the effects are coupled with the VARK implementation, there is no specific future-proofing required.

26.1 Implementation of Effects using EMCP

The EMCP is a standard RISC processor with attached DSP for fast integer multiplication. It contains a limited implementation of the VARK language for image manipulation. The subset of VARK includes compositing, convolving, filters, warps, color manipulation, tiling, and brushing. It does not include lighting effects.

The exact implementation will depend on the RISC processor chosen, although the clock speed is expected to be around the order of 48 MHz (a multiple of the PenPrint Serial Bus speed). Although VARK is processor independent, time-critical functions benefit from being specifically accelerated for a given processor. The following sections provide decomposition of an example set of VARK functions, complete with timings specific for PenPrint image parameters. See [7] for a complete definition of the VARK language, and [3] for a complete example of accelerating VARK for a specific CPU.

26.1.1 Compositing

The majority of Effects Module processing is simple compositing, involving the overlaying of a masked image over the background original image. Compositing is used by the following effects:

- borders
- characters
- captions

The process of compositing is to add a foreground image to a background image using a matte or a channel to govern the appropriate proportions of background and foreground in the final image. Two styles of compositing are supported: regular compositing and associated compositing. The rules for the two styles are:

Regular composite: new value = Background + (Foreground - Background) α
Associated composite: new value = Foreground + (1- α) Background

The difference then, is that with associated compositing, the foreground has been pre-multiplied with the matte, while in regular compositing it has not. Note that the α channel has values from 0 to 255 corresponding to the range 0 to 1.

The compositing process is memory bound, requiring a maximum of 4 memory accesses for each pixel:

- Read Alpha (α)
- Read Background
- Read Foreground
- Write Result

When the α value is 0, the background is unchanged and the number of cycles to process the pixel is 1 (read α).

For a 850 × 534 pixel × 3 color image, the total number of pixels is 1,361,700. At 4 cycles per pixel, the total number of cycles is: **5,446,800**.

If the resultant image is not stored locally, but instead transferred immediately to the Printer Module, the compositing process is less than a simplistic[1] 8 cycles per pixel transfer time, and can therefore be performed on-the-fly. It is absorbed in the transmission time and therefore effectively takes **0 seconds**.

26.1.2 Convolve

A convolve is a weighted average around a center pixel. The average may be a simple sum, a sum of absolute values, the absolute value of a sum, or sums truncated at 0.

The image convolver is a general-purpose convolver, allowing a variety of functions to be implemented by varying the values within a variable-sized coefficient kernel. In this description, the kernel sizes supported are 3×3, 5×5 and 7×7 only.

The coefficient kernel is a table in DRAM. The kernel is arranged with coefficients in the same order as the pixels to be processed. Each coefficient entry is 8 bits.

The convolve process involves 9 reads and 1 write for a 3×3 convolve, 25 reads and 1 write for a 5×5 convolve, and 49 reads and 1 write for a 7×7 convolve. For building an image pyramid (see Section 26.1.4 on page 168), a 3×3 convolve is all that is required.

The DSP allows a single cycle multiply/accumulate. The CPU must therefore generate the addresses (fixed offsets from each pixel location) and read the pixels from DRAM. This gives a minimum set of timings as shown in Table 46:

Table 46. Convolve Timings

Kernel Size	Cycles per Pixel	Cycles per 850×534 Image	Cycles per 3-color Image
3×3	10	4,539,000	13,617,000
5×5	26	11,801,400	35,404,200
7×7	50	22,695,000	68,085,000

1. 8 cycles is simplistic because it does not take any transmission overhead into account. The more likely time is 10 cycles for transmission of 8 bits over the PenPrint Serial Bus (see Section 3.1.1 on page 11). However, if the process can be performed in fewer than 8 cycles, it can certainly be performed in 10.

26.1.3 Color Substitution

It is often desirable to transform an image in terms of color. Simple color effects include removal of color to produce a greyscale image or a sepia tone image. More complex effects include exaggeration of certain colors, substitution of one color for another etc.

26.1.3.1 Simple Replace Color

One of the simplest ways to transform the color of a pixel is to encode an arbitrarily complex transform function into a lookup table. The component color value of the pixel is used to lookup the new component value of the pixel. For each pixel read from the image, its new value is read from the lookup table, and written back to the image.

The input image is in the L*a*b* color space, with the luminance channel separated from the chrominance channels. The L*a*b* color space is particularly conducive to good use of replacement of color. Examples include desaturation for greyscale (leaving L* alone and making a* and b* constant), brightening or darkening of an image, exaggeration of particular colors etc.

If the lookup table starts at an appropriate address, the whole process takes 3 cycles per pixel: one to read the old value, one to read the new value from the lookup table, and one to write the new value back to the image. The lookup table required is 256 bytes.

For a 850 × 534 pixel × 3 color image, the total number of pixels is 1,361,700. At 3 cycles per pixel, the total number of cycles is: **4,085,100**. The 3 lookup tables consume a total of 768 bytes.

If the resultant image is not stored locally, but instead transferred immediately to the Printer Module, the compositing process is less than a simplistic[1] 8 cycles per pixel transfer time, and can therefore be performed on-the-fly. It is absorbed in the transmission time and therefore effectively takes **0 seconds**.

26.1.3.2 Complex Color Transformations

Rather than perform arbitrarily complex color transformations exhaustively, excellent results can be obtained via a tri-linear conversion based on 3 sets of 3D lookup tables. The lookup tables contain the resultant transformations for the specific entry as indexed by L*a*b*. Three tables are required: one mapping L*a*b* to the new L*, one mapping L*a*b* to the new a*, and one mapping L*a*b* to the new b*. Tri-linear interpolation can be used to give the final result for those entries not included in the tables.

Tri-linear interpolation requires reading 8 values from the lookup table, and performing 7 linear interpolations (4 in the first dimension, 2 in the second, and 1 in the third). High precision can be used for the intermediate values, although the output value is only 8 bits.

The size of the lookup table required depends on the linearity of the transformation. The recommended size for each table in this application is $17 \times 17 \times 17^2$, with each entry 8 bits. A 17 × 17 × 17 table is 4913 bytes (less than 5KB).

1. 8 cycles is simplistic because it does not take any transmission overhead into account. The more likely time is 10 cycles for transmission of 8 bits over the PenPrint Serial Bus (see Section 3.1.1 on page 11). However, if the process can be performed in fewer than 8 cycles, it can certainly be performed in 10.
2. Although a 17 × 17 × 17 table will give excellent results, it may be possible to get by with only a 9 × 9 × 9 conversion table (729 bytes). The exact size can be determined by simulation. The 5K conservative-but-definite-results approach was chosen for the purposes of this document.

To index into the 17-per-dimension tables, the 8-bit input color components are treated as fixed-point numbers (4:4). The 4 bits of integer give the index, and the 4 bits of fraction are used for interpolation.

Tri-linear interpolation takes 11 cycles. Table reading can occur concurrently and takes 8 cycles. Likewise, the 1 cycle output pixel write can occur concurrently. Note that this 11 cycle time must occur once for each color, thereby totalling 33 cycles. For a 3 color 850 × 534 pixel image, the elapsed time is **44,936,100 cycles**.

For a 48 MHz processor, the entire color conversion process takes **0.93 seconds**.

26.1.4 Construction of Image Pyramid

Several functions, such as warping, tiling and brushing, require the average value of a given area of pixels. Rather than calculate the value for each area given, these functions make use of an *image pyramid*. An image pyramid is effectively a multi-resolution pixel-map. The original image is a 1:1 representation. Sub-sampling by 2:1 in each dimension produces an image 1/4 the original size. This process continues until the entire image is represented by a single pixel.

An image pyramid is constructed from an original image, and consumes 1/3 of the size taken up by the original image (1/4 + 1/16 + 1/64 + ...). For an original image of 850 × 534 the corresponding image pyramid is approximately 1/2 MB

The image pyramid is constructed via a 3×3 convolve performed on 1 in 4 input image pixels (advance center of convolve kernel by 2 pixels each dimension). A 3×3 convolve results in higher accuracy than simply averaging 4 pixels, and has the added advantage that coordinates on different pyramid levels differ only by shifting 1 bit per level.

The construction of an entire pyramid relies on a software loop that calls the pyramid level construction function once for each level of the pyramid. Note that the start address of each level of the image pyramid should be on a 64-byte boundary to take advantage of addressing mode redundancy.

The timing to produce a level of the pyramid is that for a 3×3 convolve (see Section 26.1.2 on page 166). The standard timing is 10 cycles per output pixel. For this function, we are always outputting an image 1/4 the size of the input image. Thus for a 850 × 534 image:

- timing to produce level 1 of pyramid = 10 × 425 × 267 = 1,134,750 cycles
- timing to produce level 2 of pyramid = 10 × 213 × 134 = 285,420 cycles
- timing to produce level 3 of pyramid = 10 × 107 × 67 = 71,690 cycles

Etc.

The total time is 10/3 cycles per original image pixel (generated levels of image pyramid total 1/3 of original image size, and each pixel takes 10 cycles to be calculated). In the case of a 850 × 534 image the total is 1,513,000 cycles. Multiplying this number by 3 for the 3 color channels gives a total time of **4,539,000 cycles**.

With a CPU operating frequency of 48 MHz, the timing is just less than **0.1 seconds**.

26.1.5 Warping an Image

The image warper performs several tasks in order to warp an image:

- Construct Image Pyramid (see Section 26.1.4 on page 168)
- Scale the warp map to match the image size
- Determine the span of input image pixels represented in each output pixel
- Calculate the output pixel via tri-linear interpolation from the input image pyramid

26.1.5.1 Scale warp map

In a data driven warp, there is the need for a warp map that describes for each output pixel, the center of the corresponding input image map. Instead of having a single warp map containing interleaved information, X and Y coordinates are treated as separate channels. Consequently there are two warp maps: an X warp map showing the warping of X coordinates, and a Y warp map, showing the warping of Y coordinates. The warp maps can have a different spatial resolution than the image they are scaling (for example a 32 × 20 warp map may adequately describe a warp for a 850 × 534 image). In addition, the warp maps can be represented by 8 or 16 bit values that correspond to the size of the image being warped.

There are several steps involved in producing points in the input image space from a given warp map:

- Determining the corresponding position in the warp map for the output pixel
- Fetch the values from the warp map for the next step (this can require scaling in the resolution domain if the warp map is only 8 bit values)
- Bi-linear interpolation of the warp map to determine the actual value
- Scaling the value to correspond to the input image domain

The first step can be accomplished by multiplying the current X/Y coordinate in the output image by a scale factor (which can be different in X & Y). For example, if the output image was 850 × 534, and the warp map was 85 × 54, we scale both X & Y by 1/10. This can also simply be accomplished by using the scale factors as simple deltas.

Fetching the values from the warp map requires access to 2 lookup tables. One lookup table indexes into the X warp-map, and the other indexes into the Y warp map. The lookup table either reads 8 or 16 bit entries from the lookup table.

The next step in the pipeline is to bi-linearly interpolate the looked-up warpmap values.

Finally the result from the bi-linear interpolation is scaled to place it in the same domain as the image to be warped. Thus, if the warp map range was 0-255, we scale X by 850/255, and Y by 534/255.

Table 47 lists the constants required for scaling a warp map:

Table 47. Constants Required for Scaling Warp Map

Constant	Value
XScale	Scales 0-ImageWidth to 0-WarpmapWidth
YScale	Scales 0-ImageHeight to 0-WarpmapHeight

Table 47. Constants Required for Scaling Warp Map

Constant	Value
XRangeScale	Scales warpmap range (eg 0-255) to 0-ImageWidth
YRangeScale	Scales warpmap range (eg 0-255) to 0-ImageHeight

The following lookup tables are used:

Table 48. Warpmap Lookups

Lookup	Size
X Lookup	Warpmap width x Warpmap height
Y Lookup	Warpmap width x Warpmap height

Given [X,Y] the 4 entries required for bi-linear interpolation are returned. Even if entries are only 8 bit, they are returned as 16 bit (high 8 bits 0).

Since we move along space in the output image, it is a simple matter to add XScale and YScale to the offset within the warpmaps. Transfer time is 4 entries at 16 bits per entry, with 16 bits transferred per cycle for a total of 4 cycles. This is done twice, once for each warpmap for a total of 8 cycles.

Note that all 3 colors are warped using the same warpmap, so the total warpmap calculation time is 8 cycles per output pixel. For a 850 × 534 pixel image, the elapsed time is **3,631,200 cycles**.

26.1.5.2 Calculate Span

The points from the warp map locate centers of pixel regions in the input image. The distance between the centers of the pixel regions indicates the size of the regions, and we approximate this distance via a span.

We take a given point in the warp map P1. The previous point on the same line is called P0, and the previous line's point at the same position is called P2. We determine the absolute distance in X & Y between P1 and P0, and between P1 and P2. The maximum distance in X or Y becomes the span - a square approximation of the actual shape. This is shown in Figure 123.

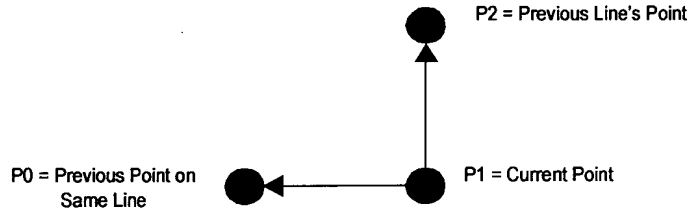


Figure 123. Calculate Span

Since we are processing the points in the sequential output order, P0 is the previous point on the same line, and P2 is the previous line's point (kept in a history buffer). P0, P1, and P2 are all 32 bit quantities.

P1 is placed in the history buffer, and taken out again at the same pixel on the following row as the new P2. Therefore 2 cycles are required: transfer of P2 from buffer, and transfer of P1 to buffer. The transfer of P1 to P0, normally a third cycle, can be masked by other activity. Since this must be done for both X and Y coordinates, the total time taken is 4 cycles.

A further 2 cycles are required for the subtraction and comparison. Since both coordinates must be subtracted and compared, this leads to 4 cycles, and a total of 8 (including history access).

Note that all 3 colors are warped using the same warpmap, so the total span calculation time is 8 cycles per output pixel. For a 850 × 534 pixel image, the elapsed time is **3,631,200 cycles**.

26.1.5.3 Calculate Pixel

We know the center and span of the area from the input image to be averaged, so the final part of the warp process is to determine the value of the output pixel. Since a single output pixel could theoretically be represented by the entire input image, it is potentially too time-consuming to actually read and average the specific area of the input image. Instead, we approximate the pixel value by using an image pyramid of the input image. Figure 124 shows the same point on two levels of an image pyramid.

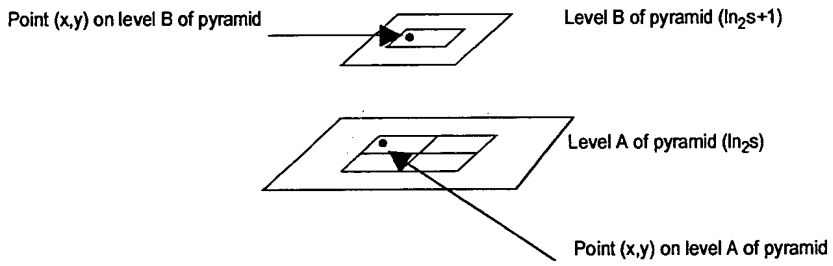


Figure 124. Corresponding Point on two levels of image pyramid

If the span is 1 or less, we only need to read the original image's pixels around the given coordinate, and perform bi-linear interpolation. If the span is greater than 1, we must read two appropriate levels of the image pyramid and perform tri-linear interpolation, as shown in Figure 125. Performing linear interpolation between two levels of the image pyramid is

not strictly correct, but gives acceptable results (it errs on the side of blurring the resultant image).

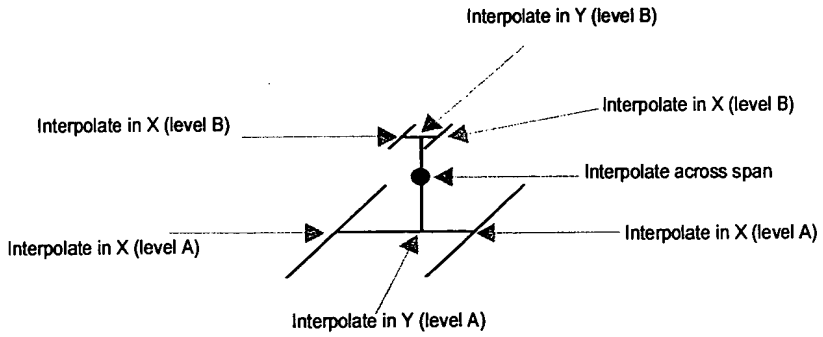


Figure 125. Trilinear Interpolation

Generally speaking, for a given span *s*, we need to read image pyramid levels given by $\ln_2 s$ and $\ln_2 s+1$. $\mathrm{Ln}_2 s$ is simply decoding the highest set bit of s. We must bi-linear interpolate to determine the value for the pixel value on each of the two levels of the pyramid, and then interpolate between them.

Tri-linear interpolation takes 11 cycles. Image pyramid transfer time can occur concurrently and takes 8 cycles. Likewise, the 1 cycle output pixel write can occur concurrently. Note that this 11 cycle time must occur once for each color, thereby totalling 33 cycles. For a 3 color 850 × 534 pixel image, the elapsed time is **44,936,100 cycles**.

26.1.5.4 Summary of Warp

The entire warp process is summarized in the following table:

Table 49. Warp Steps and Timings

Step	Base Timing	Total Timing for 3 color 850 × 534 image
Construct image pyramid	10/3 cycles per output pixel color component	4,539,000 cycles
Scale warpmap	8 cycles per output pixel	3,631,200 cycles
Calculate span	8 cycles per output pixel	3,631,200 cycles
Calculate output pixel	11 cycles per output pixel color component	44,936,100 cycles
	TOTAL	56,737,500 cycles

At a processor speed of 48 MHz, the time taken to warp a 3 color image is **1.18 seconds.**

ROM requirements for warping are directly related to the size of the warp maps, which in turn depend on the warp complexity. For simple warps, a warpmap size of **9 KBytes** is sufficient (85 × 54 × 2 coordinates × 8-bit components). For more complex warps, a warpmap size of **73 KBytes** is required (170 × 108 × 2 coordinates × 16-bit components).

References

27 References

[1] CIE, 1986, *CIE 15.2 Colorimetry: Technical Report (2nd Edition)*, Commission Internationale De l'Éclairage

[2] ISO/IEC 19018-1:1994, *Information technology - Digital compression and coding of continuous-tone still images: Requirements and guidelines*, 1994

[3] Silverbrook Research, 1998, *Artcam Central Processor: Vark Acceleration.*

[4] Silverbrook Research, 1998, *Authentication of Consumables.*

[5] Silverbrook Research, 1998, *Authentication Chip.*

[6] Silverbrook Research, 1998, *Memjet.*

[7] Silverbrook Research, 1998, *Vark Language Specification.*

[8] Wallace, G.K., "The JPEG Still Picture Compression Standard", *Communications of the ACM*, 34(4), April 1991, pp.30-44

New United States Patent Application
Priority: No. PQ0560 filed May 25, 2000 in Australia
Title: Method and Apparatus of Image Conversion
Inventor: Simon Robert Walmsley and Paul Lapstun
Assignee: Silverbrook Research Pty. Ltd.
Docket No.: PP17US